Not All Gestures Are Created Equal: Gesture and Visual Feedback in Interaction Spaces

by

Qi Yang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2015

Doctoral Committee:

Assistant Professor Georg Essl, Chair
Professor Mark S. Ackerman
Associate Professor Sile O'Modhrain
Associate Professor Gregory H. Wakefield

For mom and dad

ACKNOWLEDGEMENTS

I would like to thank my advisor Georg Essl for his unwearying support, guidance and counsel in the last five years, in addition to steering me in the right direction and tirelessly reading many of my drafts. I would also like to thank my dissertation committee members Mark Ackerman, Sile O'Modhrain, and Gregory Wakefield for their guidance. I am grateful for their rigorous, and tremendously valuable insights, which are crucial to the formation and completion of this thesis.

I would like to thank my colleagues in the sound lab which include James Juett, Sang Wang Lee, Didi Zhang, Xin Fan, and Antonio Deusany de Carvalho Junior. They were my sounding boards and mirrors, and compatriots in this journey through graduate school. Our conversations were a source of both academic enlightenment and encouragement. In particular, I owe a special thank you to Taylor Cronk for his contribution on the event-action backend in Tapperware. Many ideas in Tapperware (described in Chapter VI) also owe their inspiration to an earlier unpublished project urVen.

I would like to thank my editor Brooke Horton for smoothing out the following pages and polishing my unwieldy language. I would also like to thank Nancy Wu, Ciara Reyes, Max Radin and Mark Dong for lending me your tireless eyes for proofreading. I want to acknowledge my friends Max Radin, Yuanyuan Zhou, Jasmine Jones for bravely volunteering for our pilot studies.

I would like to thank my brothers and sisters in faith from Harvest Mission Community Church of Ann Arbor (HMCC) and the HMCC graduate student fellowship Impact for rooting and praying for me, as well as participating in some of the studies presented. A special

shoutout to Sinsar Hsie, John Wang, Victor Wong, Kevin Meng and Jeffery Yeung for being like brothers to me. In addition, I would like to thank the friends at American Friends Service Committee, my musical collaborators of the Ann Arbor Fruit Preservation Society and the University Carillon Studio. Thank you for giving me opportunities to explore my passions.

I want to acknowledge a few institutions at the University of Michigan, namely the Department of Computer Science and Engineering, the Rackham Graduate School, and the Center for Statistical Consultation and Research for all their generous support and advising. I would also like to thank the Natural Sciences and Engineering Research Council of Canada for their generous support.

Finally, I would like to give thanks to Mom and Dad. No amount of ink or characters spilled here could express my gratitude to you, and without you, none of this would have been possible. Gracias Ciara, thank you for being with me through the end of this journey.

Soli Deo gloria,

Qi Yang

May 2015

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES viii

LIST OF TABLES x

ABSTRACT xi

CHAPTER

I. Introduction 1

1.1 Thesis Overview 2
1.2 Thesis Contribution 3
1.3 Thesis Outline 5

II. Background 7

2.1 Proliferation of Gestural Interaction 7
2.2 Interactions on Modern Mobile Devices 8
2.3 Multi-Touch on Mobile 9
2.4 Feedback for Gesture Interfaces 11

III. Gesture Augmented Piano and Visualization 12

3.0.1 Keyboard as Interface 12
3.0.2 Visual Association in Gestural Interface 14
3.1 Related Works 15
3.1.1 Visualization 16
3.1.2 Evaluation Methods 17
3.2 System Implementation 18
3.2.1 Extended playing technique 20
3.3 Human Subject Study 21

3.3.1 Experiment Design . . . 22
3.3.2 Participants . . . 24
3.3.3 Results . . . 25
3.4 Visual Associations in Gesture Space . . . 31
3.4.1 Positioning the Visual in the Interactive Loop . . . 31
3.4.2 Examples of Visualization Feedback . . . 32
3.4.3 Explanation through Visuals . . . 35
3.4.4 The Purpose of Visuals . . . 37
3.5 Conclusion . . . 38

IV. Visual Programming Environment on Multi-touch . . . 40

4.1 Introduction . . . 40
4.2 Related Works . . . 41
4.2.1 Visual Programming . . . 42
4.2.2 Multi-Touch Interfaces . . . 43
4.2.3 Mobile Multi-Touch Programming . . . 45
4.2.4 Visualization for Gestures . . . 47
4.2.5 Impact of Device Size . . . 48
4.3 Representation & Interaction Design . . . 48
4.3.1 Shared Grammar . . . 49
4.3.2 Visual Representation and Interaction Modes . . . 51
4.3.3 Considerations in Touch-Based Interaction . . . 57
4.4 Experiment . . . 58
4.4.1 Design . . . 59
4.4.2 Recruitment . . . 62
4.4.3 Results . . . 63
4.5 Conclusion . . . 70

V. Fitts's Law and Occlusion on Touch Screen Drag Motions . . . 73

5.1 Background . . . 74
5.1.1 Fitts's Law . . . 74
5.1.2 Occlusion . . . 76
5.2 Experimental Design . . . 77
5.2.1 Data Collection . . . 80
5.3 Data Analysis . . . 81
5.3.1 Data cleaning . . . 81
5.3.2 Movement Time . . . 81
5.3.3 Error Rate . . . 82
5.3.4 Fitts's Law Model . . . 83
5.3.5 Time Effects . . . 85
5.4 Discussion . . . 86
5.5 Conclusion . . . 88

VI. Tapperware: Implementation 90

6.1 Overview 90
6.2 Programming Primitives 91
6.2.1 Region 91
6.2.2 Link 94
6.2.3 Group 96
6.3 Graphical User Interface 97
6.3.1 Gesture Recognition and Visualization 98
6.3.2 Utilities 100
6.4 Summary 100

VII. Conclusion 101

7.1 Contribution 102
7.2 Future Research 105

BIBLIOGRAPHY 107

LIST OF FIGURES

Figure

2.1 The on-screen software keyboard on iPad 9
3.1 Configuration of the Augmented Keyboard 17
3.2 Hand gesture recognition in Augmented Keyboard 19
3.3 Data flow of the Augmented Keyboard 20
3.4 Visual feedback generated based on hand detection 21
3.5 Musical passages used for keyboard performance study 23
3.6 Effect of learning in keyboard performance study 26
3.7 Comparison of jitter in gestures 28
3.8 Excerpts of open-ended responses from participants 30
3.9 Example visualizations in Augmented Keyboard 32
3.10 A typical strumming gesture over the harp visualization 33
3.11 Reaction of the Flock visualization to user gestures 34
4.1 The wide range of mobile touch screen device size 41
4.2 Common multi-touch gestures on iOS 44
4.3 Basic elements of the multi-touch visual programming environment 49
4.4 The interface for changing a region's texture 50
4.5 Menu representation mode 52
4.6 Creating link in menu mode 53
4.7 Icon representation mode 54
4.8 Drag gesture handle in Icon mode 55

4.9 Gesture mode . . . 55
4.10 Drag-and-drop grouping gesture in Gesture mode . . . 56
4.11 Coping gesture in Gesture mode . . . 57
4.12 Gesture-driven menu in Gesture mode . . . 58
4.13 Piano keyboard layout built with our programming environment . . . 59
4.14 Music mixer interface built with our programming environment . . . 60
4.15 Task Completion Time Wait Time in visual programming study . . . 64
4.16 Sample trace of drag gestures for one participant . . . 66
4.17 Gestures trace for all participants . . . 67
4.18 Effect of time in visual programming study . . . 68
4.19 Likert scale questionnaire in visual programming study . . . 70
5.1 Target distances and target sizes in relation to screen sizes, in scale . . . 78
5.2 Two dragging direction used in the dragging task . . . 81
5.3 Mean Movement Times of touch screen drag motions . . . 82
5.4 Error rate of touch screen drag motions . . . 83
5.5 Fitts's Law modeling using unadjusted *ID* . . . 84
5.6 Fitts's Law modeling using adjusted *IDe* . . . 85
5.7 Throughput of touch screen drag motions . . . 86
5.8 Effect of time analysis in drag motions . . . 87
6.1 Overview of the software components of the programming environment . . . 91
6.2 An urMus region with size, position and texture configured . . . 92
6.3 Tapperware `region` class . . . 93
6.4 Link class diagram . . . 95
6.5 Composition of layers in the graphical interface of Tapperware . . . 97
6.6 Gesture state machine examples . . . 99

LIST OF TABLES

Table

3.1 All mapping configurations of gestures and physical wheels used in the study 23
3.2 Classification of visualizations and the interface with respect to their functions for performer and audience. 37
4.1 List of cognitive dimension questions 70

ABSTRACT

Not All Gestures Are Created Equal: Gesture and Visual Feedback in Interaction Spaces

by

Qi Yang

Chair: Georg Essl

As multi-touch mobile computing devices and open-air gesture sensing technology become increasingly commoditized and affordable, they are also becoming more widely adopted. The expanding utility of these technologies makes it necessary to create new interaction design, specifically for gesture-based interfaces, to meet the growing needs of users. However, a deeper understanding of the interplay between gesture, and visual and sonic output is needed before meaningful advances in design can be made. This thesis addresses this crucial step in development by investigating the interrelation between gesture-based input, and visual representation and feedback, in gesture-driven creative computing.

This thesis underscores the importance that *not all gestures are created equal*, and there are multiple factors that affect their performance. For example, a drag gesture in multi-touch visual programming scenario performs differently than a similar drag gesture in a target acquisition task. The work presented here (i) examines the role of visual representation and mapping in gesture input, (ii) quantifies user performance differences and similarities in gesture input to examine the effect of multiple factors on gesture interactions, and (iii) develops tools and platforms for exploring visual representations of gestures. A range of gesture spaces and usage scenarios from continuous sound control with open-air ges-

tures to mobile visual programming with discrete gesture-driven commands was assessed. Findings from this thesis show that performance in gesture interactions is dependent on multiple interacting factors such as the mapping of gesture to sound, device size and the task scenario.

The work in this thesis reveals a rich space of complex interrelations between gesture input and visual feedback and representations, which enables both immediate design solutions and further exploration of concepts. The contributions of this thesis includes the development of an augmented musical keyboard with 3-D continuous gesture input and projected visual feedback, as well as a visual touch-driven programming environment for interactively constructing dynamic interfaces. These designs were evaluated by a series of user studies in which gesture-to-sound mapping was found to have a significant affect on user performance, along with the selection of visual representation. A number of counter-intuitive findings point to the potentially complex interactions between factors such as device size, task and scenarios, which exposes the need for further research. For example, the size of the device was found to have contradictory effects in two different scenarios. Furthermore, this work presents a multi-touch gestural environment to support the prototyping of gesture interactions.

CHAPTER I

Introduction

Personal mobile devices with touch-screens as the main interface have become commonplace and more prevalent than traditional personal computers (as of 2014, over 76% of American adults under the age of 30 own a smartphone (*Zickuhr and Rainie*, 2014)). The commoditization of open-air gesture sensing technology such as Microsoft Kinect (*Microsoft*, 2013) or Leap Motion [1] has also brought gesture interactions to new domains of applications. As a result, gesture-based input methods are becoming popular due to the widespread adoption of multi-touch and open-air gesture sensing technology in consumer electronics. Gesture-capable devices require gesture-based interaction modes, where gestures replace or complement traditional interaction methods. These new hardware developments call for new interaction designs, and also enabled the development of more *natural* user interfaces beyond the keyboard-mouse paradigm (*Buxton*, 1991).

Another trend growing in parallel to the wide adoption of gesture-capable devices is the more personal, intimate, and creative use of computing power in consumer mobile devices such as smartphones and tablets. In fact, a large portion of the popular mobile applications is creative and allows artistic expression (Instagram, Pinterest, Magic Piano [2], and etc.). The ubiquity of these platforms creates an opportunity to empower the general public. More natural interfaces, which reduce the friction of personal expression and allows

[1] `https://www.leapmotion.com`

[2] respectively `http://instagram.com`, `https://www.pinterest.com`, `http://www.smule.com/apps`

the author to have complex and interactive experiences, could be instrumental in allowing this computational power to be fully exploited.

Currently, gesture-capable consumer platforms such as mobile multi-touch devices or Kinect-like open-air gesture spaces have a gestural vocabulary limited to *tap*, *pinch*, or *swipe*. The design of more complex gestures is an area of active exploration. By replacing and/or augmenting existing interaction models with gesture, there is potential to broaden the expressive power of existing devices, thus enabling new, more personal and creative use-cases. For example, in the domain of augmented musical instruments, gesture controls and novel sensors are used to complement traditional physical controls (see *Miranda and Wanderley* (2006) for an overview). Feedback mechanisms are integral to the success of user interface. Previous works have explored many feedback methods for gesture interactions, from visual (on-screen guides, silhouettes) to tactile and audio feedback (*Charbonneau et al.*, 2011; *Sodhi et al.*, 2012; *Schönauer et al.*, 2012; *Bark et al.*, 2013). Similar to the traditional keyboard-mouse paradigm, visual feedback is most readily available for gesture interfaces due to the availability of displays (via screens or projections) and the versatility of dynamic on-screen interface elements. However, visualization for gesture interfaces is still not as well understood as more traditional interaction methods such as the keyboard and mouse. This deficient in knowledge is the guiding motivation for this thesis.

1.1 Thesis Overview

In this thesis, we investigated the interrelation between (i) visual representation and feedback, and (ii) gesture-based input across a range of interaction spaces, with a focus on the domain of creative computing. To complement previous research on the mechanics of gesture input and the design of visual feedback, we focused on the joint consideration of gesture interfaces and visualizations on multiple levels. In particular, we examined how both gestures and visuals combine to explain and guide interactions.

Visuals are essential as the primary representation of gesture interfaces and the feedback

mechanism. We studied the visual representation specifically situated in the domain of musical performance and mobile programming. We also considered a range of gesture space sizes from large open-air gestures to finer drag gesture on mobile devices. The gestures are considered in these different spaces: (i) continuous hand and arm open-air gestures in a performance scenario, where movement is mapped directly to continuous input parameters, (ii) discrete gesture commands in mobile multi-touch devices of differing form-factor, where each gesture (such as tapping or dragging) activates the corresponding commands. Within these gesture spaces and contexts, we explored how visual representation of the interface affects the user's perception, preference and performance through a series of user studies. Our work is distinctive from the works such as *Hofmeester and Wolfe* (2012) and *Bragdon et al.* (2010), which explored the role of visuals in a generic context in revealing and explaining gesture interactions to users.

1.2 Thesis Contribution

Our central research question is:

Which aspect(s), if any, of visual feedback and sonic mapping affects the efficacy of gesture-based interaction in creative computing?

This concentration leads to the following specific goals:

1. Examine the role of visual representation and mapping in real-time continuous gesture-to-input-parameter interaction, and discrete gesture-driven commands.

2. Quantify user performance in gesture input across a range of interaction spaces to examine the effect of multiple factors on gesture interactions.

3. Develop tools and platforms for exploring visual representations of gestures.

The main finding of this thesis is that *not all gestures are created equal*. We found that not only is the performance of gesture input affected by factors such as the mapping of gesture to

sound and device size, the effects are also dependent on the condition and task. The specific contributions made in this thesis are as follows:

Open-air gestures versus physical interface in musical performance

We evaluated open-air gestures versus physical wheel control in the context of a prototype gesture-augmented musical keyboard instrument. Our real-time musical performance user study showed that the selection of gesture mapping is crucial for performance and expression, and that users reported greater enjoyment using the gestural interface thani the traditional physical wheel controls. They also reported similar or higher capacity for expressiveness when using the gestural interface. This study is detailed in **Chapter III**.

Visualization design for open-air gesture performance

We developed a series of visualization examples to explore the role of visuals in real-time gesture performance using the augmented keyboard instrument described above. We developed a framework for explaining the mechanics of the interactions by varying the physical placement and temporal-causal relations between visuals and gestures. This is described in **Section 3.4**.

Gesture and visual representations in mobile visual programming

We examined the effect of different styles of visual representation between gesture-based interfaces and more traditional text and icon based interface, as well as device size with a touch-driven visual programming environment. We conducted a user study and found significant differences in user preferences and performances, with respect to the visual representations and also device form-factor. Differences in device sizes affected performance in an unexpected way: smaller devices with more restricted screen space leads to better performance on visual programming tasks. To the best of our knowledge there are no existing comparative user studies on visual programming on multi-touch devices. This is detailed in **Chapter IV**.

Device size, occlusion on drag gesture motor performance

To further investigate the effect of device size and occlusion on mobile multi-touch devices, we present a follow-up user study to quantify these effects on touch-screen dragging motions in **Chapter V**. We found that drag motions can be modeled by Fitts's Law and that device size and possible occlusion significantly affects performance. Based on the previous study, we hypothesized that devices with smaller screens would allow users to perform better. However, we found that smaller screens lead to significantly slower drag motion, possibly due to a difference in perceived user confidence. We also found that more occlusion in the task condition leads to faster performance with a lower accuracy.

Framework for prototyping multi-touch gestures on mobile

We present *Tapperware*, a multi-touch gestural environment to support prototyping of gesture interactions, which enabled the mobile visual programming study mentioned above and expounded upon in **Chapter IV**. Based on an existing cross-platform audio programming framework urMus(*Essl*, 2010b), we built a visual environment where different multi-touch gestures can be specified through a state machine, as well as modularized architecture that allows visual feedback to be developed separately. The implementation details of this framework are discussed in **Chapter VI**.

1.3 Thesis Outline

The organization of this thesis is as follows:

Background

Chapter II provides an overview of the previous work in areas of gesture interactions, the role of various feedback mechanisms for gesture interactions, and multi-touch interactions on mobile devices.

Gesture Augmented Piano and Visualization

Chapter III details the investigation of a prototype open-air gesture-augmented keyboard instrument. We present the implementation detail of this prototype as well as a user study to compare different gesture mappings to traditional physical controls. Later in this chapter, we show a series of visualization examples and a framework for developing visuals within the same prototype instrument.

Gestures in Visual Programming Environment on Multi-Touch

Chapter IV presents a visual, touch-driven gestural environment for constructing dynamic interfaces. The same chapter also details the effect of varying visual representation and device form-factor on user performance based on a user study examining visual programming.

Touch-screen Drag Motion Study

Chapter V presents a follow-up user study to the previous chapter to quantify the effects of occlusion and device size on touch-screen dragging motions, using Fitts's Law of human movement as a model.

Implementation of the Visual Programming Environment

In **Chapter VI** we describe the system architecture of Tapperware, a multi-touch visual programming environment. We show how it can be used for prototyping multi-touch-driven interfaces.

Conclusion

In **Chapter VII** we summarize the contribution of this thesis to the current field of work and discuss directions for future research.

Portions of **Chapter III** have been published in *Yang and Essl* (2012), *Yang and Essl* (2013), and *Yang and Essl* (2014). Parts of **Chapter IV** have been accepted at NIME15, and we plan to submit results of the user studies in **Chapter IV** and **Chapter V** for publication in the near future.

CHAPTER II

Background

Gesture-based input is becoming popular due to the widespread adoption of multi-touch and open-air gesture sensing technologies. Mobile devices with touch screens as the main interface have become common-place and the commoditization of open-air gesture sensing technologies such as Microsoft Kinect (*Microsoft*, 2013) or Leap Motion [1] has opened a new domain of motion based interactions. These new hardware developments call for new interaction designs beyond the keyboard and mouse in traditional PCs. While the adoption of these technologies is recent, there is a rich body of work in the past several decades on gesture interaction, multi-touch interaction as well as visualization as a feedback mechanism in interfaces.

2.1 Proliferation of Gestural Interaction

Gesture-based input is increasing in popularity as an alternative interaction model for computing devices. The adoption of affordable smartphones and tablets is outpacing traditional personal computers (*Milanesi et al.*). At the same time, popular gaming systems are making more use of gesture as part of their interface (e.g. Nintendo Wii, Playstation Move and Microsoft Kinect). These new trends have led to a new generation of users who, having grown up using these computational devices instead of traditional PCs, are more accustomed to

[1] `https://www.leapmotion.com`

gesture being part of the interaction vocabulary. In some cases gesture-based input has also supplanted the traditional mouse-keyboard and physical controllers.

2.2 Interactions on Modern Mobile Devices

On personal mobile devices such as smartphones and tablets, the gesture on multi-touch screens has become the standard interaction method. Interactions on these devices are generally characterized by:

- Portability (screen size of $4'' - 10''$)

  The portability of the device enables more ubiquitous usage in contrast to traditional desktops or laptops. This further encourages more personal use-cases, such as entertainment or creativity. The wide range of screen sizes also potentially enables or hinders different types of usage.

- Multi-touch capable touch screen as the primary user interface

  The lack of a physical keyboard or other physical buttons for primary interaction presents a unique interaction model where almost all interactions occur via touch. Unlike a physical keyboard, the virtual on-screen elements, such as buttons or sliders, can be changed dynamically through software.

The smaller size of these devices means that even the largest at $10''$ is significantly smaller than desktops ($20''$ or larger) or laptops ($13''$–$15''$), which have more than twice the area. Furthermore, most touch screen mobile devices do not have a physical keyboard and rely on a virtual on-screen keyboard for text input. When activated, the software keyboard interface takes up further space on the screen (Figure 2.1), allowing for even less screen space for software interfaces.

These factors limit the efficacy of using traditional software UI, since restricted screen size and lack of a physical keyboard make it difficult to use the keyboard-mouse-driven

interfaces found on a PC. The touch-driven hardware interface is an opportunity for new interaction design paradigms based on gestures.

The portable nature of mobile devices encourages new use-cases that are more personal than traditional computing devices. The class of popular mobile applications (e.g. social networks, photo-sharing applications such as Instagram[2], or creative games such as Magic piano[3]) points to the general public's growing interest in creativity and personal artistic expression. Given their growing ubiquity over PCs (as of 2014 over 76% of American adults under the age of 30 own a smartphone (*Zickuhr and Rainie*, 2014)), mobile devices are well-positioned to potentially further empower the general public in the domains of personal expression and creativity. However, their computational power has yet to be fully exploited.

Figure 2.1: On typical tablet such as the iPad, the on-screen keyboard takes almost half of the screen space when activated, leaving only 54% of the space for applications.

2.3 Multi-Touch on Mobile

Multi-touch interactions differ from the traditional keyboard and mouse model in several ways. For example, on a desktop computer, the keyboard is usually used for text input, and the mouse (and arrow keys on a keyboard) is used for precise selection of on-screen elements such as buttons and menu items. However, on a multi-touch mobile device, both tasks are done using touch. Compared to the mouse, touch input has performance advantages as a pointing device in accuracy and throughput of target acquisition (*Sasangohar et al.*, 2009)

[2] `http://instagram.com`
[3] `http://www.smule.com/magicpiano`

and when bimanual interaction is preferred (*Forlines et al.*, 2007). However, in addition to occlusion by the user's hand or finger (*Nacenta et al.*, 2009) (or the "fat-finger" problem (*Cockburn et al.*, 2012)), when finger tips are used for selection or pointing, the precision of touch is lower than a mouse cursor. As a result, interactive regions on a touch screen have to be larger than on a desktop to provide targets that are easy to hit accurately (*Vogel and Baudisch*, 2007). The average index finger is between 16–20mm wide (*Dandekar et al.*, 2003). Previous work and developer guidelines have recommended 22mm as the minimal diameter for touch targets. This minimum size varies depending on the spacing between the targets (*Hall et al.*, 1988; *Scott and Conzola*, 1997). Recent developer guidelines have recommended a minimum of 9mm (*Apple Inc.*, 2013; *Google Inc.*, 2013).

These differences mean that the traditional mouse-based user interface cannot be ported to touch devices directly. For example, in a text editor, both text selection and movement of the text entry cursor will be difficult since touch input is less accurate at pointing at a precise location. The finger used for pointing will also obscure the text content that the user needs to see. This is a known problem and solutions such as *Vogel and Baudisch* (2007) have been proposed. As a result, touch interfaces have increased the size of interactive elements such as buttons and controls, exacerbating the limitation of smaller screens.

In the case of text entry, typing speed and accuracy afforded by virtual software keyboards on touch screens also suffers compared to physical keyboards. The typing speed on a full-sized mechanical keyboard averages roughly 60 words-per-minute (WPM), or higher with experience (*Grudin*, 1983; *Roeber et al.*, 2003). *Chaparro et al.* (2010) found that the average typing speed on iPad's virtual keyboard (which is the same size as a physical keyboard) is significantly slower at 42 WPM, with similar error rates (1–2%). On smaller touch screen devices (screen size smaller than 7″), the on-screen virtual keyboard is significantly smaller than the regular physical keyboard, and as a result typing with all fingers becomes difficult or impossible. On most phones for example, screen sizes range from 3.5″ to 4.5″, and the virtual keyboards are designed to be used with just two thumbs. Since the on-screen key-

board is software based, touch zone location and size usually has to be predictively adjusted according to language models, and most devices autocorrect the user's input to reduce errors. The average typing speed on a mobile phone touch screen is roughly 30 WPM (*Goel et al.*, 2012).

Most consumer touch-based software and operating system vendors also make use of gestures such as tap and hold, swipe and pinch/stretch in addition to on-screen buttons that respond to a simple tap. However, these novel gestural interaction methods are criticized due to their poor visibility, discoverability, learnability, and consistency (*Norman and Nielsen*, 2010). These differences mean that the traditional mouse-based user interface cannot be ported to touch devices directly, and new gesture-centric user interface is needed.

2.4 Feedback for Gesture Interfaces

The feedback mechanisms for these gestural interaction methods remain an open space with a range of approaches from on-screen guides or silhouette (*Charbonneau et al.*, 2011), on-hand projections (*Sodhi et al.*, 2012), to wearable tactile feedback (*Schönauer et al.*, 2012; *Bark et al.*, 2013). Despite these works, which we will discuss in more detail in **Chapter III** and **IV**, the visualization of gesture interfaces is still not as well understood as traditional interaction methods such as the keyboard and mouse, and there remains space for exploration.

To fully exploit the power of gestural interaction methods, Donald Norman called for the development of a standardized convention for gestural interfaces, so that the visualization would "follow the basic rules of interaction design". This would take into account explicit feedback to aid the discovery of possible gesture commands and to explain the dynamics of their execution (*Norman*, 2010).

CHAPTER III

Gesture Augmented Piano and Visualization

Musical keyboards are musically expressive and are well suited for discrete note performance. However, smooth adjustments of performance parameters that are important for digital synthesizers or samplers are difficult to achieve. Since the 1970s, such adjustments have often been achieved using pitch and modulation wheels at the left side of the keyboard. Contemporary gestural sensor technology makes it increasingly easy to offer continuous inputs, and gestures' potential for expressivity makes it an ideal candidate for music performance. We augmented the musical keyboard with a 3D gesture space using the Microsoft Kinect (*Microsoft*, 2013), an infrared based depth camera for sensing and top-down projection for visual feedback. This interface provides 3D gesture controls to enable continuous adjustments to multiple acoustic parameters such as those found on the typical digital synthesizers. Using this system we conducted a user study to establish the relative merits of free-hand gesture motion versus traditional continuous controls. We also explored the design space of potential visual feedback for this open-air gesture interface.[1]

3.0.1 Keyboard as Interface

The popular piano-style musical keyboard enables the player to address multiple discrete pitches concurrently and directly. In contrast, wind instruments produce a single pitch at a

[1] Content of this chapter has been published in *Yang and Essl* (2013, 2014)

time and require complex chorded fingering. Further, in string instruments such as violin or guitar, polyphony is limited by the number of strings, and by the geometry of the hand that provides the fingering. Also, the initial activation and reactivation of notes on a keyboard does not require preparation such as stopping the strings or activating multiple valves on a wind instrument.

Despite the ease of keyboard playing, it does came with drawbacks. After the onset of each note, the player has limited control of the quality of the sound. This is in contrast to bowed or wind instruments, which have a range of expressive timbre controls after the onset of each note. In the case of the traditional piano, limited timbre controls are provided by pedals to dampen of the strings and therefore the amount of sympathetic resonance between strings.

The pipe organ does offer means of timbre control through knobs or tabs, commonly referred to as organ stops. The player pushes or pulls on the stops to discretely activate or mute different sets of pipes, changing the timbre of the sound produced by actuating the keys. Pipe organs have developed a wide range of timbres that are enabled by different combinations of pipes, but the physical interface has seen little change, as the the stops are not designed for timbre changes while keys are being held down (more recent pipe organs allow configurations to be saved in advance and loaded during the performance), while the crescendo and swell foot pedals provide limited continuous timbre controls.

Digital synthesizers, sampler instruments and MIDI controllers usually feature a keyboard for pitch selection and note activation. For parameter adjustment during live performance, they traditionally feature one or two wheels (or in some cases joysticks) next to the keyboard for modulation or pitch bending control. We want to see if open-air hand gestures provides better means of adjustment during live performance.

It is easy to perform continuous gestures using hand motions in space, hence they make a good candidate for real-time continuous timbre control especially in improvised music. At the same time, gestures of musicians can be musically expressive and aesthetically pleas-

ing, even when not instrumental in producing the actual sound. The expressivity of gestural control makes it a natural fit in music domain and the utility of capturing these gestures for musical performances is recognized (*Rovan et al.*, 1997). Theremin, an early purely gesture-based instrument, uses antennas to sense hand position, also requires a high level of skill to play (*Paradiso*, 1997). Gesture controls are used often for other theremin-like music instruments or to augment traditional instruments (*Wanderley and Depalle*, 2004). More recently, Kinect offers affordable 3D sensing to be used to build gesture-based interfaces for music (*Yoo et al.*, 2011; *Berg et al.*, 2012), and for augmenting acoustic instruments (*Odowichuk et al.*, 2011).

Our prototype system uses an off-the-shelf depth camera to track a range of hand motions, positions and gestures in real-time, making it suitable for live performance and the goals of this paper. The sensing of position and hand-width creates a space with multiple continuous degrees of freedom, allowing multiple parameters to be controlled simultaneously. The gesture space also allows either hand to be used for hand gesture controls, in contrast to the fixed location of pitch and modulation wheels on the left of a standard MIDI keyboard.

3.0.2 Visual Association in Gestural Interface

In a gestural digital instrument such as our gestural-augmented keyboard, we can arbitrarily configure the relationship between input and output. Nothing in computation requires one choice over another. This in principle leaves it open how to choose a mapping between input and output. However this choice of mapping is what in various ways defines the instrument. This is a canonical problem in new music instrument design known as the "mapping problem" (*Miranda and Wanderley*, 2006).

An important part of the mapping problem relates to our natural experience of acoustic instruments. An acoustic instrument "explains itself" to the performer and the audience. It enable the performer to associate the act of initiating sound by physical interactions until

these actions become muscle memory, and performance becomes intuitive. This is facilitated by the pure physical interfaces and how sound are produced in these instruments. The act of pressing a physical key or blowing into a wind column is directly associated with the initiation of the sound, as are the actions that affect the timbre during the sound production, such as varying the pressure and speed of bowing on a string instrument. Hence acoustic instruments tend to suggest a kind of causation that can be consistently experienced and learned. In this sense a new music instrument should strive to explain itself to both the performer and the audience.

Using the augmented keyboard prototype, we also explored the question of visuals as part of the gesture interaction loop to aid this explanation. We explored possible choices and functions of visual in this setup using some examples we constructed, and suggest some broader views from these perspectives.

3.1 Related Works

Our system draws on the augmentation of established traditional musical instruments, and continuous controls for musical instruments with gestures. Both fields have both extensive prior works and we refer the reader to comprehensive reviews (*Paradiso*, 1997; *Miranda and Wanderley*, 2006).

How to best support continuous control in conjunction with the keyboard interface is a longstanding problem and has seen many proposals. When designing the first hard-wired commercial analog synthesizers, Bill Hemsath in collaboration with Bob Moog and Don Pakkala invented the pitch and modulation wheels (*Pinch and Trocco*, 2004), which became the canonical forms of continuous control on electronic keyboard interfaces ever since. Early analog synthesizers had many continuous controls via rotary potentiometers and sliders, but in many canonical cases the pitch and modulation wheels were the only ones that survived the transition to digital synthesizers. However, continuous control in keyboard performance remained an important topic. *Moog* (1982), later with collaborators Rhea (*Moog*

and Rhea, 1990) and Eaton (*Eaton and Moog*, 2005) experimented for decades with prototypes to add continuous control to the surface of the keys themselves. This idea has also been explored by *Haken and Tellman* (1998), *Lamb and Robertson* (2011), *McPherson and Kim* (2010) and *McPherson* (2012).

Another idea that has been proposed is the augmentation of the action of the key itself. The classic aftertouch, where extra levels of control are available once the keys are fully depressed, is an early example of this (*Paradiso*, 1997). Precise sensing of key position can be achieved through various means such as optical interruption sensing (*Freed and Avizienis*, 2000). More recently, *McPherson and Kim* (2011) described the augmentation of traditional piano keys through a light-emitting diode sensing mechanism that is capable of inferring performance parameters from the key action.

3.1.1 Visualization

The relationship between sound and visual display has taken a central place digital music instrument design. The work of Sergi Jorda (*Jordà*, 2003) on tangible interface and Golan Levin's work on shape based projection interaction serve as examples that inspired the way we attack these questions. Levin's Manual Input Workstation (*Levin and Lieberman*, 2005) most immediately inspired our thinking, using real-world physical metaphors to "explain" the gestures. He used camera combined with overhead projection to construct a shape-based performance system that included sound.

More broadly, similar setup are used on large-surface multi-touch displays, as pioneered by *Han* (2005). *Davidson and Han* (2006) demonstrated the use of virtual control elements (onscreen sliders, knobs) for sound synthesis interface. In an effort to reveal the mechanics of a digital instrument to audiences, many musicians use video projection of the instrument, or crafted visualization such as ROUAGES (*Berthaut et al.*, 2013).

Closest in setup to our system are *Takegawa et al.* (2011) who added top-down projection to a musical keyboard in order to provide score visualizations, and *Rogers et al.* (2014) which

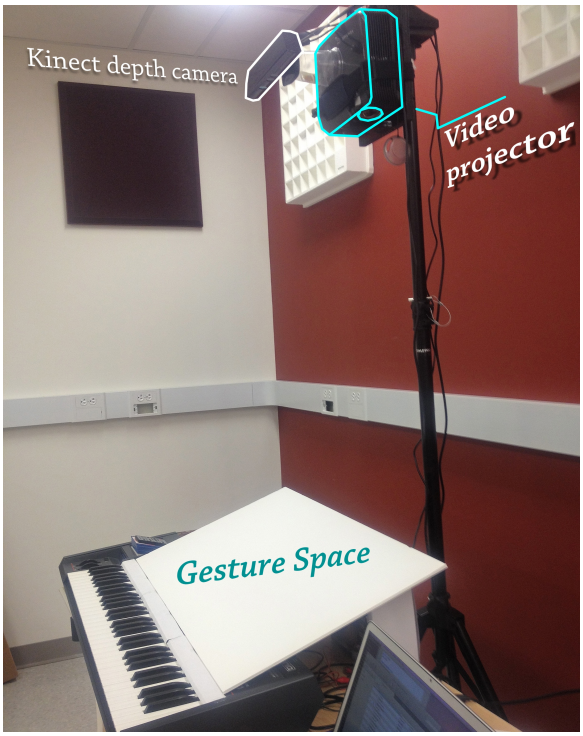

Figure 3.1: Configuration of the Augmented Keyboard

uses the same projection for performance feedback and pedagogy, but neither make use of open-air gestures.

3.1.2 Evaluation Methods

In addition, literature on evaluation methodologies exist for designing digital music instruments. Notably *Wanderley and Orio* (2002) suggested using musical tasks and adapting human computer interaction methodologies for evaluating input devices in the area of music instrument evaluation. *O'Modhrain* (2011) proposed a framework where the roles and goals of different stakeholders (such as the audience, performer, manufacturer and etc.) of the musical instruments are all considered for the evaluation of instrument designs. *Jordà* (2004) proposed a measure of efficiency of musical instruments based on the expressive power and diversity, and complexity of the input interface. Our evaluation draws ideas from *Wanderley and Orio* (2002) by using HCI performances metrics of input devices with a well-defined musical task.

3.2 System Implementation

Our system uses a Kinect depth camera and a video projector installed above a MIDI keyboard, facing down towards the keyboard (Figure 3.1). The Kinect depth camera, projector and keyboard are connected to a single computer which processes the sensor data from Kinect and MIDI data from the keyboard, while controlling a software synthesizer to produce the sound. A white projection surface placed above the keyboard allows a clear view of the projected visual feedback.

The Kinect depth camera is used to capture three-dimensional data on the gesture space, in the form of an 11-bit monochrome, 640×480-pixel video stream sampled at 30 Hz, with the brightness indicating the distance from the camera. This video stream is passed through background and noise removal and fed into a blob-detection algorithm using OpenCV (*Culjak et al.*, 2012). The blob-detection algorithm used was based on labeling pixel contour components (*Chang et al.*, 2004). Using the initial keyboard setup as a background, the image with the background removed is passed through blob-detection and we can then detect the presence and position of the player's arms as they enter the gesture space. We isolate the player's hand positions by capturing the extremity of their arms, and we use the centroid of their hands as the position. Using the center of their hand as reference, we also measure the distance to the camera, which in this case corresponds to the height of the hand. (See Figure 3.2 for the stages of processing depth camera data) At the same time, we can also compute the width of the hands to see if they are open or closed. The hand motion trajectory inferred from this position is past through an averaging filter of five frames to remove the jitter caused by the noise in the depth camera.

Using the Processing framework (*Reas and Fry*, 2006) as a bridge, the hand position data is mapped to timbre control MIDI messages to be sent to a software synthesizer (Figure 3.3). MIDI note pitch and attack velocity messages from the keyboard are also sent to the synthesizer. We also use Processing for visual feedback (Figure 3.4), which is projected unto the surface beneath the gesture space. The detected location of the player's hands is

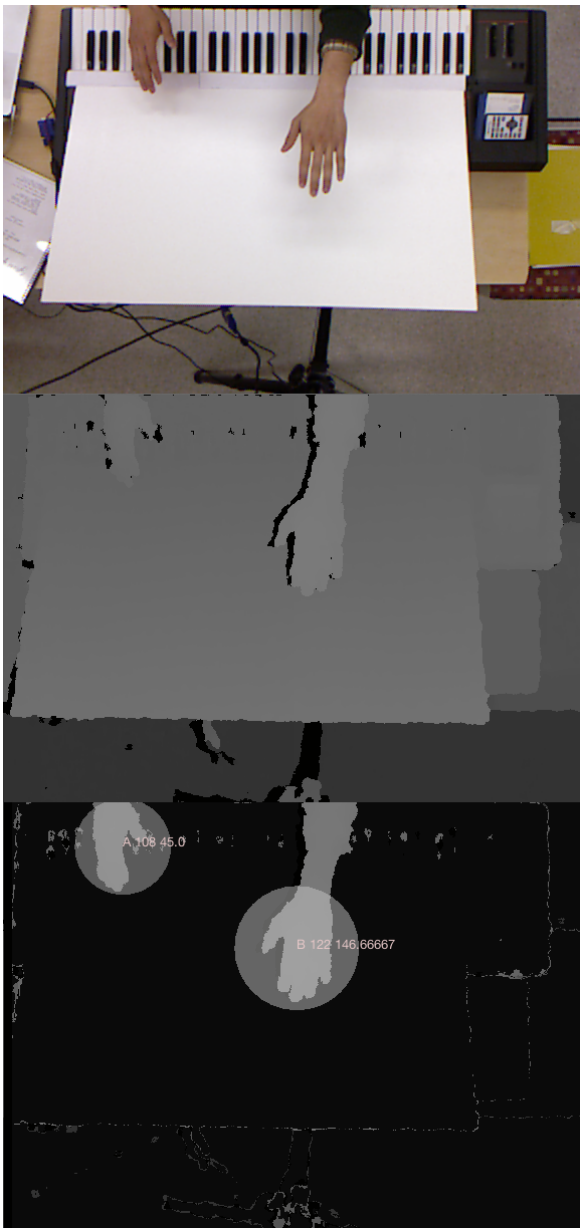

Figure 3.2: Kinect video stream, depth camera stream, and image after background removed with hand position derived from blob-detection.

displayed, as well as vertical and horizontal bars showing the gesture axes that are currently active and their current values, and circles showing the size of the palm as well as the height of player's hands.

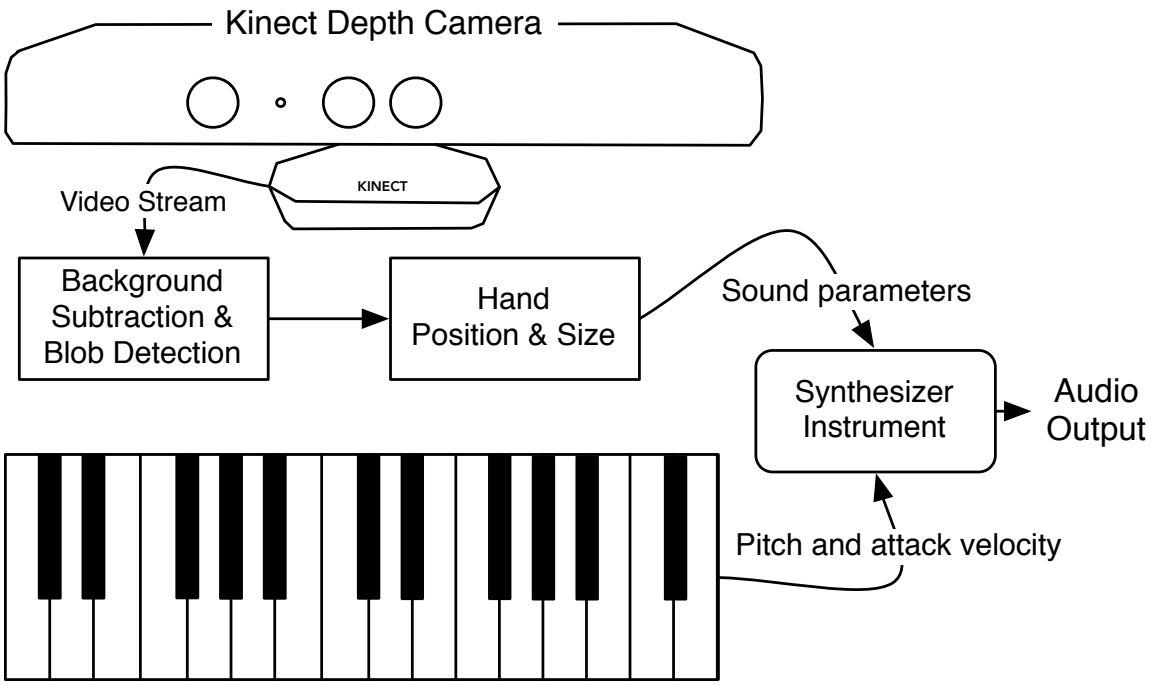

Figure 3.3: Data flow of the Augmented Keyboard

Overall the latency in the system from Kinect sensor to displaying visualization and MIDI control messages is estimated to be 174ms, with a standard deviation of 23ms, less than the 33ms it takes for Kinect sensor to refresh. (Note that latency measurements were conducted after a forced operating system update and may not fully reflect original user study)

3.2.1 Extended playing technique

With our system, a keyboard player can play normally using both hands on the keyboard, just as any traditional keyboard. For continuous gesture controls, the player can move either hand into the gesture space immediately above and behind the keyboard, while using the other hand to continue playing simultaneously. The gesture space can also be configured to

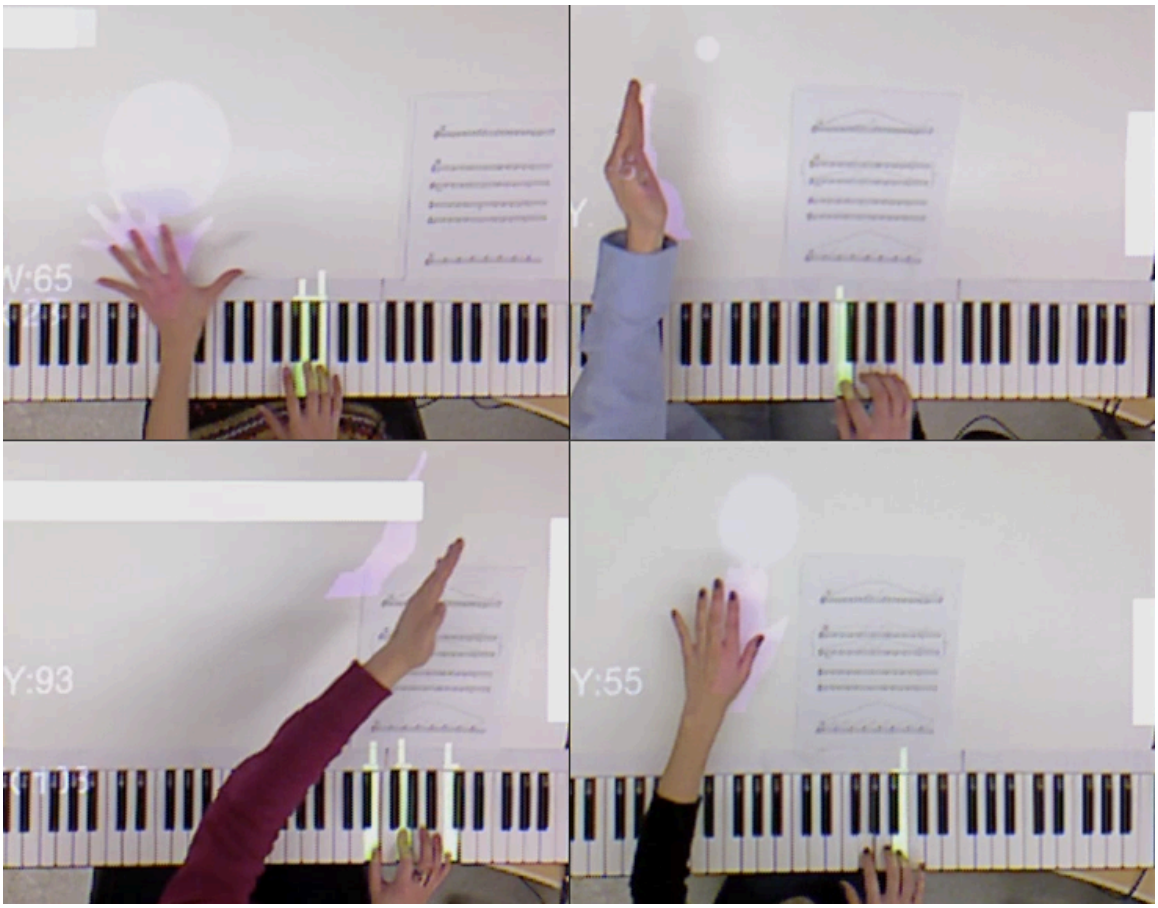

Figure 3.4: Visual feedback generated based on hand detection

be directly above the keys on the keyboard itself, so any wrist motion or other hand gesture during normal playing can be captured and used for continuous control.

3.3 Human Subject Study

We conducted a user study to evaluate how our system performs versus the physical controls featured on conventional electronic keyboards. In addition, we wanted to examine the mapping between gesture types and timbral parameters, as well as to study ergonomic issues such as fatigue, learnability, and enjoyment.

3.3.1 Experiment Design

Our study consisted of two parts: a 45 – 50 minute playing session on the augmented keyboard, and an exit questionnaire.

To test continuous timbre manipulation after onset, we asked each participant to play three simple passages of monophonic melodies and chords on the keyboard, which require only a single hand to play. At the same time, the participant was to move the other hand in the gesture space to control one or two parameters of the synthesizer that affect the timbre of the sound produced.

We chose a low-pass cutoff filter (henceforth "filter", for brevity) and a tremolo (an oscillation in amplitude but not pitch) effect to be applied to a generic synthesizer sound. The two effects were chosen because they have distinct timbral effects even when applied concurrently. A musical score of the passage is provided (see Figure 3.5), with timbral effects marked as curves above the notes, with vertical position showing the amount of the effect. The filter effect is notated as a slowly increasing or decreasing timbre change, while tremolo are notated as a gradual increase to the maximum with a sharp cutoff soon after.

For comparison, we chose three distinct gestural axes to map to the two effects, as well as two physical wheel controls on the electronic keyboard. We detected the left-right movement of the player's hand (X), the front-back movement (Y), and the width of their hand (W, which changes when the hand is opened or closed, or alternatively when the wrist is turned). For physical control, we detected the pitch bend wheel (wheel1) and modulation wheel (wheel2) on the keyboard. These were then mapped to one or two timbral effect parameters. Similar to most MIDI keyboards, on the keyboard used for the experiment the pitch bend wheel is spring loaded, while modulation wheel is not, and zero timbral effect is always mapped to the neutral position on the spring-loaded wheel.

We tested all combinations of mapping one or two gestures to one or two effects using a full factorial design. We did the same with mapping physical wheel controls to effects, in total with ten configurations of control scheme mapped to a single effect, and eight config-

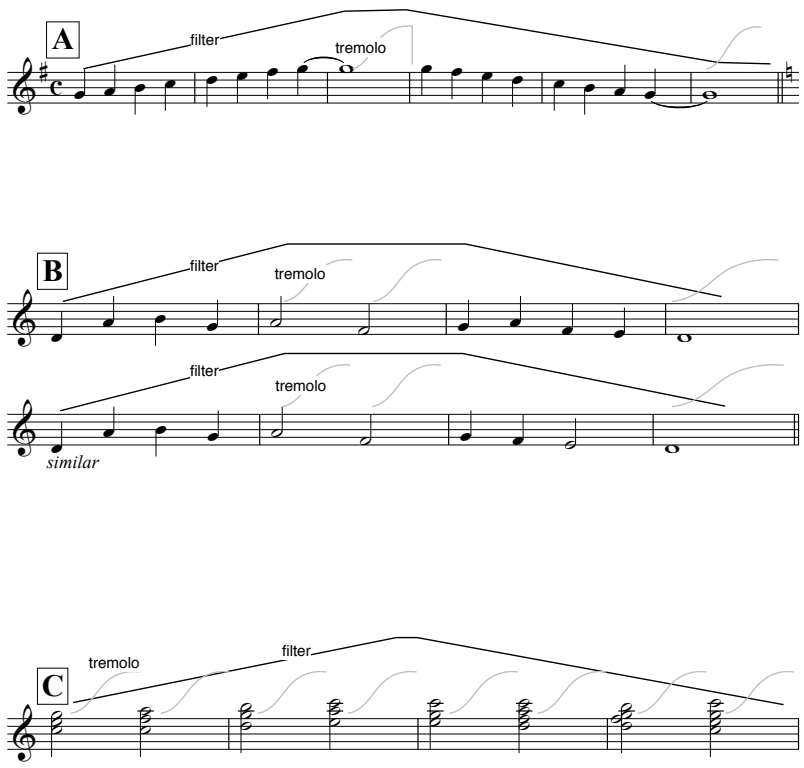

Figure 3.5: Musical notation of timbral effects used for our study. Three passages of varying difficulty are used.

urations of two controls mapped to two effects (See Table 3.1).

Config.	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Filter	Y		W		Wh1		Wh2		X		Y	W	Wh1	Wh2	X	Y	X	W
Tremolo		Y		W		Wh1		Wh2		X	W	Y	Wh2	Wh1	Y	X	W	X

Table 3.1: All mapping configurations of gestures (X,Y,W) and physical wheels (Wh1-pitch bend, Wh2-modulation) to the two effects (low-pass filter, tremolo). Each column is one configuration, empty cell indicates that the effect is not used.

At each session, the participant was asked to fill out the screening survey. After a learning period of about 5 minutes to play the passages without using any timbral effects, the configurations are presented. Since we recruited experienced piano players, the initial learning period of 5 minutes allows them to learn to play the simple passages fluently (which only requires a single hand). Due to the length of each playing session, we anticipated that

not all participants are able to complete all 18 configurations. As a result we only presented all configurations without the X gesture in randomized order first, then if there was time remaining, the configuration containing X gestures were presented in randomized order afterwards. During the actual study, out of the 22 participants, only two were unable to complete all configurations, but we kept the partially randomized presentation order for all participants for consistency.

For each configuration the participant was given one to two minutes to learn to play the passage with notated timbral effects, and then play one last time after the participant indicated that they feel ready to play, where their performance was recorded. This procedure was repeated for all three passages. New configurations were introduced without pause after each one was finished.

Although our system makes no distinction between the left and the right hand, for consistency the participants were asked to use their right hand for playing the melody and left hand for timbre controls. After completing all the configurations, the participant was invited to improvise timbral effects on music of their choosing, or to play one of the test passages using their own timbral effects, using a control configuration of their choosing. Then they were asked to fill out the exit questionnaire. In the questionnaire we use five-point Likert scale questions to assess, for each configuration, ease of learning, expressiveness, fatigue, fun, and personal preference. We also used the ISO 9241-420 questionnaire (ISO, 2011) to evaluate potential discomfort.

3.3.2 Participants

We recruited undergraduate and graduate students and faculty members at the University of Michigan. 22 participants participated in the study, with 45% female. 80% participants are between the age of 19 – 25. All had keyboard instrument experience, with more than 80% having five years or more experience , and one third of them currently studying music at the college level. Participants were compensated for their time.

3.3.3 Results

We recorded MIDI performance data from the keyboard for each configuration, as well as MIDI controller messages from the mapped gestures or physical controls. We then used this to compute task completion time, error, and smoothness of continuous controls, which will be discussed below.

3.3.3.1 Task Completion Time

We measured the time each participant took to play each passage for the final time after one or two practices. Based on observation participants encountering difficulties playing with hand gestures stuttered or paused more often, and were likely to take longer than the normal tempo they established during the practice phase. Task completion time can capture performance degradation due to cognitive load, motor performance difficulty and other related performance characteristics. Hence it serves in the author's view as a useful measure of performance competence.

We discarded data from 5 participants due to technical problems in recording data. After running a two-factor ANOVA on the task completion time of single-effect configurations (where one control is mapped to a single effect), we found that the completion time has high variance overall, and that neither controls or effect types have a statistically significant ($p > 0.05$) effect on the task completion time.

When two gestures or physical controls are mapped to two effects simultaneously, we found that passage A and B exhibited no significant difference between different control type and parameters. This is likely attributable to the fact that in none of two passages do two parameters need to be adjusted concurrently (Figure 3.5), one parameter only need to be held at a constant value while the other is adjusted. For passage C we found controls have significant effect ($F = 3.7, p < 0.0178$) on completion time. T-tests show that in particular the combination of X-filter and W-tremolo or Y-tremolo are better than many physical wheel configurations ($t = 4.11, p < 0.0008$, we used $p < 0.05/N$ after Bonferroni

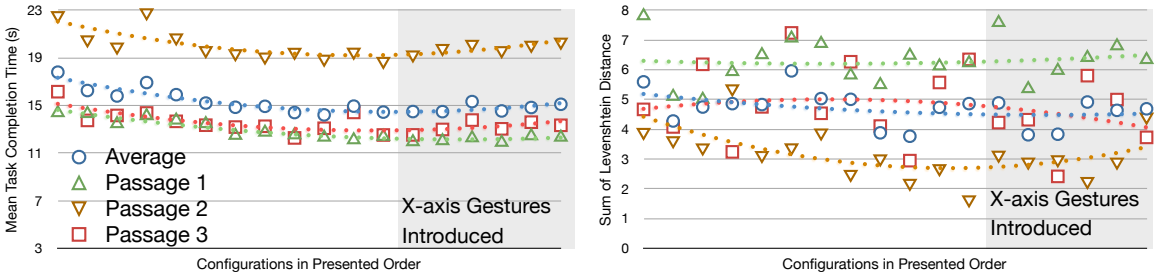

Figure 3.6: Learning curves with polynomial curve fit, with some effect in task-completion time, little effect in edit distance

multiplicity correction as threshold of significance). X-filter and W-tremolo combination is also significantly better than X-tremolo and W-filter ($t = 4.19, p < 0.0007$), with no other configurations showing significant differences. This likely because this passage requires two parameters to be adjusted concurrently.

Although many of the configurations that use X-axis are better than physical wheels, we cannot claim significance, since X gestures were confounded by not being fully randomly presented with other mappings. The measured effect could be explained in multiple ways; one possible explanation is improvement over time.

We investigated this possibility by inspecting progression of task completion time chronologically in the order of presentation (See Figure 3.6). The curve does show a slight learning effect during the first ten configurations presented. After that, before the X gestures are introduced in the last six configurations, there is little improvement. In fact, the increase in time for passages *after* the first ten configurations is a counter-indication for X gestures being confounded by learning effects, suggesting that advantage of X gestures over physical controls may be a real effect. However this is not conclusive, as the slight increase at the end can also suggest fatigue after playing for about 35 minutes.

3.3.3.2 Levenshtein (Edit) Distance

We adopted Levenshtein distance (*Levenshtein*, 1966), an algorithm to compute the minimal difference between two strings in terms of basic edit operations, as a measure of the errors

participants made during playing. Similar to task completion time, errors may correspond to difficulty in performing the continuous timbral effects. For each recorded performance, we compare the MIDI note data with a gold standard performance derived from the score. Each passage is considered as a sequence of notes, and the Levenshtein distance between the recording and "gold standard" is computed, as the number of mistakes (missing a note, inserting an extra note, or playing the wrong note) the participant made.

Since participants performed many passages with few errors, the data is sparse, and some passages have no errors at all. We aggregated errors from all three passages, a two-factor ANOVA shows no strong effect in either control schemes used or the effect mapped to. Similar to task completion time, there are no significant difference for single-effect configurations. In the case of dual-effect, X-filter and Y-tremolo performed significantly better than Y-tremolo and W-filter configuration and one physical wheel configurations ($t = 3.58, p < 0.0028$), with no other significant differences.

Similar to task completion time, we examined the possible effects of presentation order on Levenshtein Distance. We found no clear effects of learning; only Passage 2 shows some effects of presentation order (Figure 3.6). The absence of clear effects in Levenshtein distance after the first eight configurations further supports the possibility that the advantage of X gestures may be real.

3.3.3.3 Continuous Control Smoothness

We analyzed the MIDI controller data derived from either the hand motion or the physical wheels, to measure the smoothness of the continuous controls. Jitter in control (manifested as fluctuation in the controlled parameter) suggests possible difficulty in operating the control, or stumbles when the participant is confused by the mappings, or fatigue. As the participants are told to make timbral effects gradual and smooth as notated, the presence of unintended jitters should reflect the quality of the performance.

The MIDI controller data are sampled at roughly 25 Hz and have a resolution of only 7

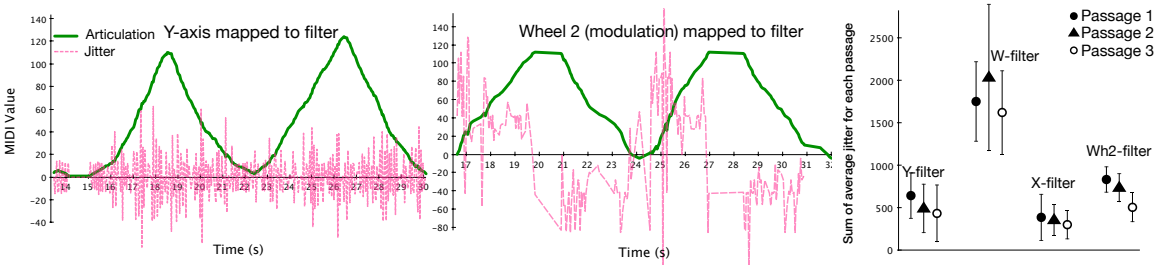

Figure 3.7: Jitter in typical (a) gesture and (b) physical wheel controls. Jitter, computed as numerical second derivative, is scaled down by a factor of 100 to fit visually. (c) Wheel control exhibit significantly more jitter.

bits (128 discrete values). To measure the jitter in the continuous controls, we use standard three-point numerical differentiation to estimate the second derivative of the effect values, to measure changes in acceleration. By cursory observation, the MIDI controller data derived from the Kinect sensor have a significant amount of noise, even after the necessary smoothing (See Figure 3.7), while physical wheels exhibit no noise when they are not actuated by the player.

Due to technical problems, we only recorded and analyzed the gestures and modulation wheel mapped to filter for nine participants. Comparing only jitter in single-effect configurations, an ANOVA shows the control scheme to have a significant effect ($F = 31.5, p < 0.000001$), and W gestures have significantly more jitter than all others ($t = 3.97, p < 0.0063$, see Figure 3.7), X gestures have less jitter than using modulation wheel ($t = 4.81, p < 0.0019$), with no other significant differences. Given that the Kinect sensor is generally noisier than physical wheels, the advantage of gestures producing continuous timbral effects with less jitter is significant. Our experiment setup did not have control to account for noise in the potentiometer in the wheels versus optic/vision sensing, however we do observe that the wheels have no noise when it is not being moved. It should be noted however, that since W gestures exhibit more noise, the difference cannot be due to sensor noises in physical wheel controls.

3.3.3.4 Exit Survey

After participants completed the playing session, they were asked to fill out an exit survey consisting of five Likert scale questions for each configuration they played, ISO9241-420 Assessment of Discomfort, and open-ended questions for feedback. Due to the large number of configurations tested, we asked participants to evaluate discomfort of gesture controls in comparison to physical wheels in general.

We analyzed the five-point Likert scale questionnaires using the pairwise Mann-Whitney U (MWU) test. The MWU only shows significance for dual-effect configurations, with gestures being easier than physical wheels ($U = 100, p < 0.0392$). Within gestures, W-tremolo is easier to learn than W-filter ($U = 94, p < 0.0245$). Most configurations are easy to learn. On expressiveness, participants responded that single-effect configurations were less expressive than dual-effect ($U = 86, p < 0.04257$). Within dual-effect configurations, using physical wheels were worse than some gestures ($U = 105, p < 0.0367$), with no other significance. When asked if the configuration was fun to play, 57% responded fun, and 11% not fun. Multiple effects were always more fun than single effect, regardless of the control scheme ($U = 82, p < 0.04426$). In addition, dual-effect configurations with W-tremolo were more fun than other configurations ($U = 113, p < 0.0226$). When the participants were asked to rate based on personal preference, the MWU shows W-tremolo to be least preferable among single-effect configurations ($U = 102, p < 0.02994$). However, for dual-effect configurations, W-tremolo is preferable to configurations where other gestures are mapped to tremolo.

For ISO9241-420 assessment of discomfort, participants were asked about fatigue of gestures versus physical wheels in general. The gestures are considered better in terms of force required, smoothness, accuracy and general comfort, with no significant differences in other factors. There is a clear tradeoff between finger and arm fatigue, with physical wheels causing more finger fatigue, while gestures cause more arm fatigue. No significant differences in fatigue are found between each configuration.

On the last open-ended question, participants mentioned that gestures improve expressiveness, and are fun to play (See Figure 3.8). They also mentioned that taking one hand away for timbre control limits the complexity of the music that can be played, and causes more fatigue. Subjects describe gesture controls as "natural" or "fluid", but also stated that different mappings can be confusing to learn, especially in the short time given. Although our system has an estimated latency of 174ms, only one participant mentioned that the system can be "it is slightly unresponsive", likely due to the latency.

Enjoyment	Expressivity	Fatigue
"the sound is definitely fun!"	"Allows more expressivity with the gesture controls than with the mod wheels."	"it is slightly unresponsive distracting to music reading, and uncomfortable (especially with the wrist)"
"It is much more fun!"	"I felt I had more direct control over the expression of the music."	"My only concern is that playing for hours could get extremely tiring."
It was fun however, doing two at the same time may get confusing, especially switching through them so fast.	yes there's more flexibility in movement with gestures. i guess you can say it's more expressive as well.	"for extended periods of times it is very tiring, making the mod wheel much more practical."
	"it limits playing to one hand."	
	"only one hand is taken for dynamics such that both hands cannot be used to play the piano keyboard."	
	"the articulations don't make up for loss of a hand in playing."	
	"it feels more like conducting and allows for more natural dynamic expression"	
	The gesture control is more fluid, but it does require some getting used to.	
	"the significantly better control over the variations in sound than the mod wheel."	

Figure 3.8: Excerpts of open-ended responses from participants

3.3.3.5 Summary

Objective metrics (task completion time, Levenshtein distance, jitter) that measure the participant's performance with the system suggest that, when multiple parameters are controlled concurrently, there are advantage in using gestures over physical wheels, as long as the gesture mappings are chosen well. The difference is however insignificant when only

a single effect is mapped or when two parameters are not adjusted concurrently. We also found some gesture mappings perform better than others, particularly when W is mapped to tremolo in any dual-effect configurations, suggesting that the action of opening the hand or turning the hand to affect W may be a good match to the tremolo effect. The results from subjective surveys agree with this finding. The subjective surveys also show that participants find the augmented keyboard generally fun and expressive, and there is a tradeoff between finger and arm fatigue caused by performing continuous timbral effects, depending on whether gestures or physical wheels are used.

While the initial 5 minutes of learning period for the musical passage familiarized the participants to play the passages fluently without using gestures or physical wheels (all are experienced piano player), the learning period for each gesture/physical wheel configuration is brief. As a result, the participants' experience with novel gesture or physical wheel augmented playing is low in comparison with their expertise with keyboard. The results presented here reflect the performances of trained keyboard musicians using a novel augmented keyboard instrument, without extensive training on the instrument. This might not be indicative of the performance of someone who had extensively studied this particular instrument.

3.4 Visual Associations in Gesture Space

Using the same system, we also explored some possible choices and functions of visuals in the gesture space with some examples of different visualization metaphors (Figure 3.9) and suggest some broader views from these perspectives.[2]

3.4.1 Positioning the Visual in the Interactive Loop

In our gesturally augmented keyboard, one or more performers, as well as the audience (possibly through mirrored and magnified projection) can have access to the projected vi-

[2]Content of this section has been published in *Yang and Essl* (2013)

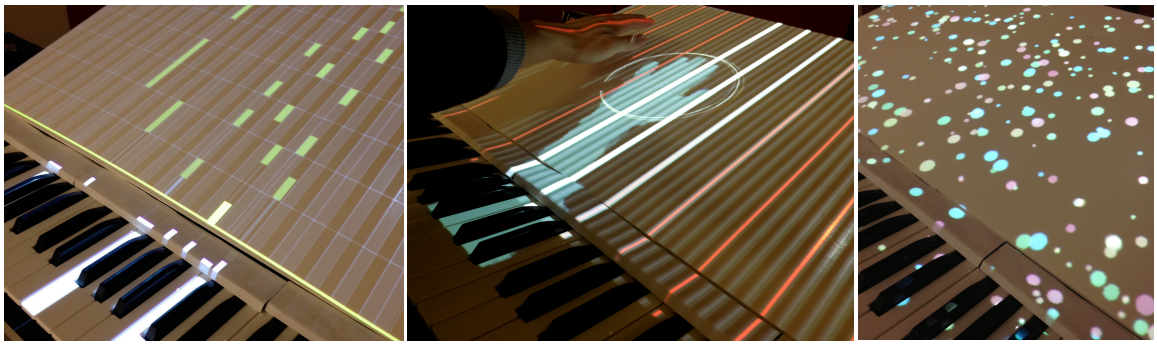

Figure 3.9: Implemented example visualizations: (a) Piano-Roll, (b) Harp, and (c) Flock

sual, which is situated in the same space as the performative gestures that are recognized. Beyond the simple mapping of continuous hand motion to single axis parameter control as discussed earlier, what is the function of the visualization in such an interactive performance system?

In this setup we have a range of modalities that make up the performance, including visual output through projection, sound produced as part of the interaction and multiple modes of control. In our case, the control is a combination of discrete control through key actions, and multi-dimensional continuous control in the gesture space.

When defining an instrument one could think of the process as trying to construct a kind of "meaningful" relationship between input and output components for the performer and the audience. How this is perceived may well differ depending on the role of the observer, with respect to the nature of active engagement with the instrument. This definition of meaning is a difficult open problem to which we make no claim of providing a solution. Rather, what we want to discuss ways to reason through the impact of choices made on a number of concretely implemented examples.

3.4.2 Examples of Visualization Feedback

We implemented three visualization examples for the instrument (Figure 3.9). They are a piano roll display, a harp-like interface, and a flocking display.

The piano roll display is implemented as an animated waterfall notation showing notes

to be played which fall down towards the musical keyboard. The performer is expected to play the indicated notes when their visual representations fall "onto" the keyboard, while the keys are gradually lit as the notes move down closer to the keyboard, indicating a need for the performer to prepare. This is a common form of score visualization for performance or pedagogy, such as in musical games (e.g. Rock Band[3], Dance Dance Revolution[4] and Rocksmith[5], Magic Piano by *Hamilton et al.* (2011)) or in learning (*Rogers et al.*, 2014).

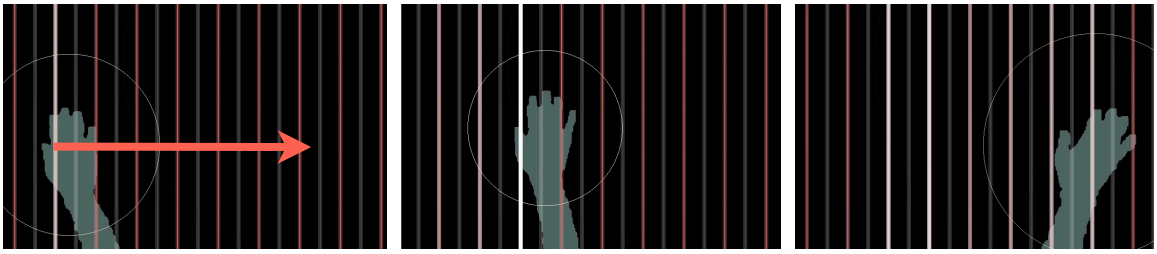

Figure 3.10: A typical strumming gesture over the harp visualization

In the harp visualization, the keyboard is used for selecting pitch classes to be activated, in a similar fashion as pedals on harps mute and unmute sets of strings. When the performer presses and holds down keys, the corresponding pitch classes are activated, and a visual representation of the set of strings for these pitch classes are shown via projection, but no sound is produced at this stage. To play the actual notes, the performer can wave his or her hands in the gesture space while the "strings" are activated, and the corresponding note is played each time the performer's hand moves across the string's location, producing an arpeggio of notes similar to strumming a traditional harp. The general gesture of the harp performance with this visualization is depicted in Figure 3.10.

In the Flock visualization, the triggering of the notes is further separated from the direct input of the performer. The music keyboard is used to select a set of pitches, which are assigned to individual entities simulated and visualized as a flock of particles moving organically on the projection surface, emulating a school of fish or other small aquatic life. Without gesture input the particles will simply move randomly and no sound is produced.

[3] http://www.rockband.com/
[4] http://www.ddrgame.com/
[5] http://rocksmith.ubi.com/

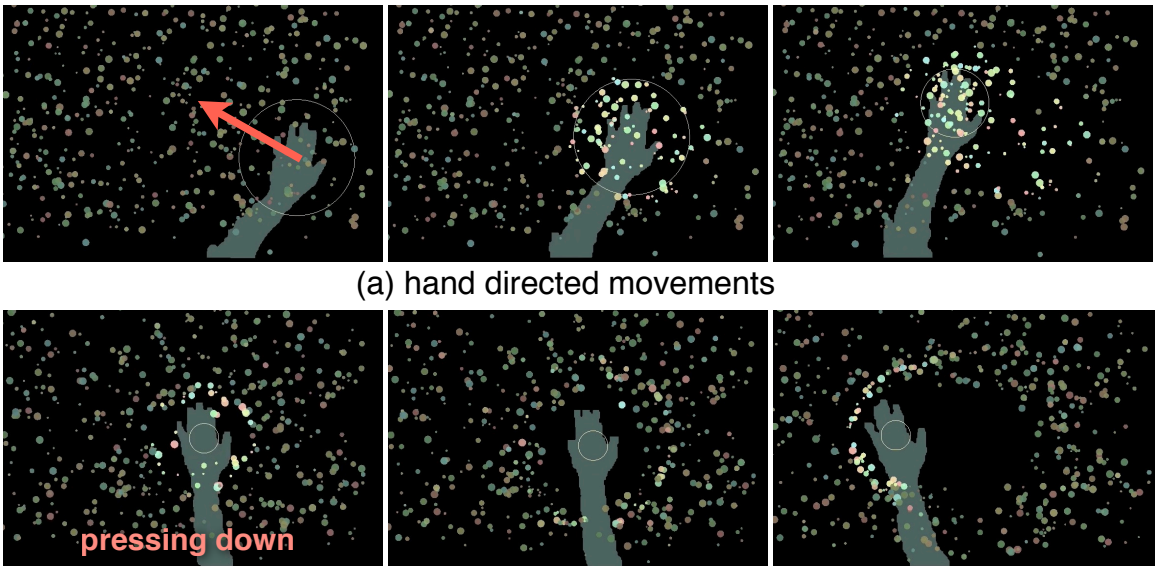

Figure 3.11: Reaction of the Flock visualization to user gestures

When the performers extend their hands in the gesture space, the movement of the particles can be directed by moving their hands faster than a threshold velocity, at which point the particles will try to follow the rough direction of the movement. When the performers press down to touch the projection surface, their hands repel particles nearby, causing them to quickly move away. Details of these interaction modalities can be seen in Figure 3.11.

The movement of each particle is used for sonification, as its assigned pitch is sounded whenever its velocity is above a threshold. As the performer uses gesture to manipulate the flock, the sound is produced dynamically as part of the particles' reaction to the gesture inputs, corresponding roughly to the overall level of activity in the simulated "fish-pond".

These examples contain a number of design choices that are varied along certain dimensions. The degree of directness differs, from very high in the case of the piano roll, to rather displaced in the case of the flock. Another important difference is the locus of pitch activation. In the piano roll example, pitches are selected traditionally by pressing keys. In the other two examples the locus of pitch selection is moved towards the gestural space.

Some of these examples also suggest a future outcome of the performance. For example the piano roll will suggest the correct notes to be played when the visual entities hit the bot-

tom of the display. The harp also suggest an anticipated outcome as specific pitches are be pre-selected even if the respective virtual strings have not yet been activated. With the flock example it is less clear if a future outcome can truly be anticipated from the visualization.

3.4.3 Explanation through Visuals

What does the visual do to explain how the performance functions, and how the visual is likely going to be perceived?

Within the context of articulating the role of perception in instrument design, enactive principles have become important in articulating the need to be conscious of the role of the action in perception (*Essl and O'Modhrain*, 2006). Gaze can be seen as an activity itself, and while looking is rather passive ultimately, there are a few notions that come to the fore more strongly when recognizing activity in visual perceptions.

One of these is attention. In our setting we are interested in how visual cues in the interface suggest where the attention of the performer and the audience should be. The site of performance can be quite complex given our interface design. It can be the keyboard or the gesture space, or a complex joint configuration between the two. What the audience should pay attention to may well define how the instrument is understood.

An important concept here is causation. What is perceived to be the main causative event that triggers sounds? It is sensible to view the perception of causation as emerging from a set of gestalt principles (*Wagemans et al.*, 2012). Common-fate and co-occurrence of percepts across multiple modalities suggest a common causative process. For example, if a gesture that looks like an impact is followed by an impact-like sound, the gestalt of the setup can lead to a perception of a causation of the sound.

Further, the perceived locus of causation may impact the notion of attention. The mechanisms of shifting attention between perceptual modalities are still not fully understood, through some progress has been made (*Bonnel and Haftser*, 1998). However it is known that audio can shift attention and hence direct the observer's gaze in a certain way and

in turn again reinforce the perceived causation of the performance. However, the visuals themselves can also direct attention and hence suggest the locus of causation.

To make this more explicit, we view the function of the visualization with respect to the actual locus of triggering sounds. To this end let us distinguish between two actions in musical performance, pitch selection and temporal sound triggers. Pitch selection is the activity of defining which pitches will be activated in performance. Temporal sound triggers are events in time that actually trigger pitches. In a typical acoustic instrument these two are by physical necessity co-located.

In our three examples, the visualization serves to direct and reinforce the locus of pitch selection and temporal triggers in different ways. In the case of the piano roll display, the locus of pitch selection and temporal triggers are co-located in the keys just like one would expect from a traditional piano performance. Further the visualization reinforces and points to this locus of performance by having its display moving towards the locus of performance.

In the case of the harp, the locus of pitch selection and of temporal trigger is split. The temporal triggering happens through gesture over the gesture space over the projected visuals. The pitch selection happens through the physical keyboard. The locus of causation is in the virtual plucking of the strings in the continuous gesture space. The visualization suggests the progression of the causal chain from pitch selection at the keyboard to temporal selection by plucking, by highlighting the selected pitch classes in response to changes of keys on the keyboard, and animates when each virtual string is "struck". The attention is not only sonically but visually drawn to the strumming part of the harp performance.

Finally, in the case of flock we again have a split between selection and temporal trigger. The keys select different sonic outcomes by priming the pitches that will be played. However there is no visual representation of this in the display. The display shows the flocking behavior which, when interacted with, will cause temporal sound triggers. Hence the interface strongly suggests the locus of attention to be drawn to the gesture space only, and de-emphasize the percept of the key selection, although the key selection of course remains

visible to both the audience and the performer.

		Piano Roll	Harp	Flock
	Directness	Yes	Yes	No
Output	Anticipation	Yes	Yes	No
	Attention	Visuals Keys	Gesture Visuals (Keys)	Gesture Visuals
	Causation	Keys	Gesture	Gesture
Input	Sound Trigger	Keys	Gesture	Gesture
	Pitch Selection	Keys	Keys	Keys

Table 3.2: Classification of visualizations and the interface with respect to their functions for performer and audience.

3.4.4 The Purpose of Visuals

These dimensions discussed previously allow us to construct a space of factors for each visualization in Table 3.2. The individual components of this space interact in non-trivial ways. For example, we argued that Sound Triggers are important for establishing the site where the causation of the performance is going to be perceived. This in turn will direct the attention to that locus.

More importantly the space of factors allows us to articulate the purpose of the visualization in the following manner. The bottom two rows of the design space, Sound Trigger and Pitch Selection can be viewed as input to the performance. Whereas Anticipation and Attention relate to where an observer is pulled in terms of visual and auditory cues, which are the output of the system. Hence we get at varied purposes of the co-location or lack thereof between visual attention, auditory attention and site of input. This can be understood in terms of how the performance will function for either a performer or the audience.

To illustrate this, let us contrast the case of the Harp against the Flock example. Both use the same input mechanisms. Pitch is selected via the keyboard and sounds are triggered by hand gestures. However the ways they are perceived are different. The Harp example

provides direct cues that link the selection process of the keys to the visualization. Hence one can expect some attention to be potentially directed at the key play. However, in the case of the Flock example all the visual feedback is designed to focus the attention of the audience to the gesture space with no visual cues about the keyboard. Furthermore, the visuals do not anticipate any particular action. Hence the key input disappears as a factor in all the output characteristics.

In addition, the real-world metaphors borrowed by these two examples inform the performer and the audience's expectation in terms of directness of the interaction. Either of the Harp and Flock example have photo-realistic visuals, instead use animations such as the activated and strummed string and the reacting movements of the flock to explain the interaction. The string animation reacts immediately when the sound is triggered by gesture, reinforcing the directness of the interaction, while particle movements in Flock is more sustained and has longer delay in reaction to gesture input, corresponding to the low immediacy.

3.5 Conclusion

We augmented the musical keyboard with a gesture space, using a depth camera for sensing and top-down projection for visual feedback of gestures. Through an user study we found that improved performance is dependent on the particular mapping between gesture and sound effect. This suggests that the choice of mapping is critical. As an example, using a change of hand width for tremolo effect shows significant improvement in performance compared to traditional pitch wheels as well as other mapping. Testers also reported that gestures have similar or improved expressivity over physical wheels for multi-parameter controls.

The same sensing and visual feedback setup has a wide range of potential applications, including supporting other styles of playing using multiple interaction modalities or pedagogy. We discussed ways to reason through the function of visualization in this setup by

means of notions of attention, causation, and anticipation among other factors. We examined three examples of visualization with this process and highlighted how this helps reason through the impact and meaning of the visualization. Different dimensions of the role of visualizations form a space of classification that illustrates the relationship between inputs and perceived outcome, and how visualization help revealing the mechanics of the gesture interface.

CHAPTER IV

Visual Programming Environment on Multi-touch

4.1 Introduction

Recently there have been efforts to make mobile computational power more accessible through building programming languages for mobile touch screen devices (*Essl*, 2010a). Many approaches take cues from the extensive previous work on visual programming for desktop PCs (due to limited space here we refer to more comprehensive overviews such as *Shneiderman* (1983); *Green and Petre* (1996); *Johnston et al.* (2004)).

Multi-touch interface for mobile touch screen devices is different in many aspects from traditional mouse and keyboard input, and require a rethinking of the interaction design. For example, the pointing accuracy of the finger requires larger tap targets (*Lee and Zhai*, 2009) and is further complicated by the occlusion of the operating hand or finger (*Nacenta et al.*, 2009). In addition, there is also a wide variation of device sizes to be considered, from a typical size of 5" on phone to 10" on tablet (see Figure 4.1)

Although there has been some work on tasks-based assessments with touch screen sizes (such as *Chae and Kim* (2004); *Oehl et al.* (2007); *Raptis et al.* (2013)) looking at general tasks such as web browsing and information seeking, to the best of our knowledge, visual programming tasks and interaction representations on multi-touch devices have not been examined before. We explored this design space by building a visual environment where simple dynamic interfaces can be interactively assembled, and then evaluated three different

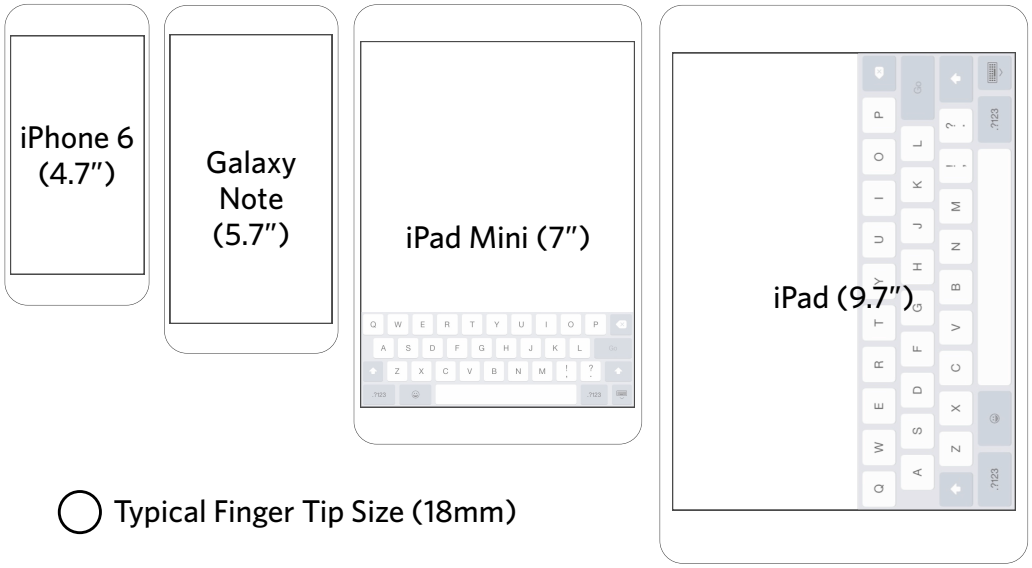

Figure 4.1: The wide range of mobile touch screen device size, ranging from 4 – 5” on typical mobile phones, to 7” or 10” on tablets

visualization and interaction modes using the same underlying visual grammar, on different-sized tablet devices.

We expected to see larger screen lead to better performance given the complexity of the tasks. However, we found that smaller tablets have higher efficiency than larger tablets based on timing measures. We also found that prior experience with tablet interaction is not a significant factor, while prior programming experience is. The results on interaction modes are complex and may vary depending on the task.

4.2 Related Works

The design of our prototype draws from the following areas: visual programming languages and interaction design for multi-touch devices. The latter is focused more specifically on programming on mobile multi-touch devices.

4.2.1 Visual Programming

Visual programming language and environment have been explored extensively with the prevalence of graphical user interfaces on PCs with keyboard and mouse input. This is often done with the intention of exposing the potential of computation to a wider audience (known as end-users) by making programming more accessible.

The cognitive benefit of visual representation for programming and problem solving was recognized early (*Larkin and Simon*, 1987). Making the text-based programming visual can reduce the initial learning curve by avoiding the need to memorize textual syntax or exact keywords. Multiple approaches to visual programming languages (including Boxer by *diSessa and Abelson* (1986), HANDS by *Pane, Myers, and Miller* (2002), and Kodu by *MacLaurin* (2011)) have used principles of Human Computer Interaction (HCI) to guide the design of the programming interfaces.

Among visual programming languages, visual blocks are often used to represent program logic and to allow users to manipulate and combine these blocks spatially to form programs (e.g. Scratch (*Resnick et al.*, 2009) and Hopscotch (*Leavitt et al.*, 2013)). Another approach is data-flow programming language, which both visually and conceptually makes use of the metaphor of nodes and edges (e.g., LabView (*Hils*, 1992), or computer vision (*Reimer et al.*, 2011)) (*Johnston et al.*, 2004). Data-flow metaphor is also used for programming interactive media, such as sound/video processing or event-driven programs. The languages or interfaces are generally designed for domain experts in the specialized context of programming for music or interactive media (e.g., Pure Data (*Puckette et al.*, 1996) and MAX/MSP (*Puckette*, 2002)).

Although text-based programming languages still dominate the landscape of commercial software development, various commercial framework and integrated development environments also incorporate visual interfaces for programming user interface or building

prototypes. Visual programming tools such as Quartz Composer[1], Form[2] or Unreal Blueprints [3] are some examples that make use of flow-based metaphors for programming.

The visualization of programming is also important in the field of live coding, a musical performance practice where a programmer writes code using a specialized, interpreted language in a concert setting to generate and perform impromptu music. Some live coding languages are designed with a predominantly visual component (*Blackwell and Collins*, 2005; *McLean et al.*, 2010), since live coding performances usually need to show a lay audience the programming process, and also convey what the programmer is doing.

The design of our visual environment draws from a number of established visual programming representations and underlying mechanisms, from data-flow to interactive or live programming (as there is no distinctive mode between editing and running the program). Our aim is to investigate the effect of different visual representations on the performance of multi-touch gesture interactions while the "grammar" of the language is held constant, so as not to focus on the particular choices in constructing the underlying programming concepts.

4.2.2 Multi-Touch Interfaces

Touch screens as an interface have been explored since the 70s (*Buxton*, 2010). One of the earliest examples of multi-touch interfaces is demonstrated in *Krueger et al.* (1985)'s VIDEOPLACE. The technology for building multi-touch screens has became commercially viable (*Buxton*, 2007) and is widely adopted on mobile phones and tablets, generating interest in Mobile HCI research on touch screen interfaces.

Previous work found that direct touch interfaces have benefits beyond the traditional indirect and text-based interaction using mouse and keyboard (*Forlines et al.*, 2007; *Terrenghi et al.*, 2007; *Sasangohar et al.*, 2009). Interacting by directly touching the interface elements

[1] https://developer.apple.com/library/mac/documentation/GraphicsImaging/Conceptual/QuartzComposerUserGuide/qc_intro/qc_intro.html

[2] http://www.relativewave.com/form

[3] https://docs.unrealengine.com/latest/INT/Engine/Blueprints

instead of indirectly through a pointing device, has been found to be more efficient for bimanual tasks (*Forlines et al.*, 2007). *Ishii and Ullmer* (1997) developed prototypes for tangible interfaces where users interact directly by hand with digitally augmented physical objects instead of onscreen interfaces.

Touch screen interfaces can be seen as a natural extension from the direct manipulation interfaces (*Shneiderman*, 1983) developed with pointing devices, such as the mouse. Sketchpad (*Sutherland*, 1964) is an early instance of this, which uses a pen to draw and manipulate onscreen objects. Although not physically tangible, touch screen interactions can have some aspects of tangibility. The co-location of the user's hand or finger and the visual interface on a touch screen, combined with the use of physical metaphors and tightly coupled feedback can form an experience where virtual onscreen objects can tangibly respond to touch manipulations.

Existing work on interface design for multi-touch interaction has many approaches. Common approach in consumer software is the adaptation of keyboard-mouse-driven interfaces for the finger, which leads to larger touch targets (e.g., larger buttons and different placements to adapt to position and occlusion of hand (*Biegel et al.*, 2014)). Commercial software for multi-touch still relies on visual buttons, but is starting to unlock the power of gesture interfaces with more complex gestures than "tap and hold", "swipe" or "pinch" (see Figure 4.2). For example, Apple's iOS running on iPad uses a multi-finger swipe gesture to activate the multi-tasking interface or switch between appli-

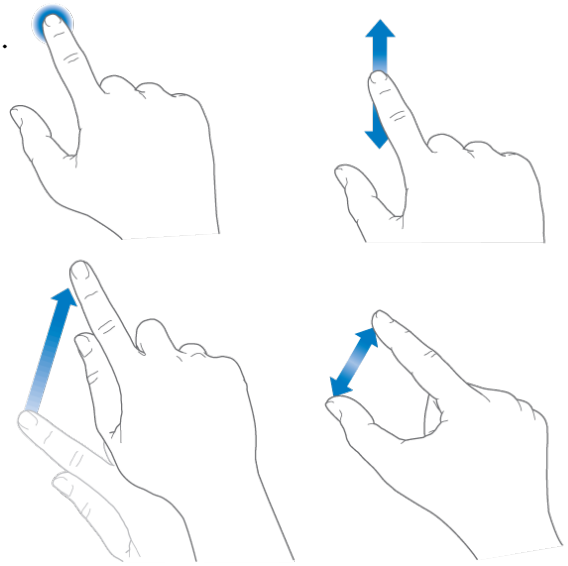

Figure 4.2: Common multi-touch gestures on iOS: "tap", "drag", "swipe", and "pinch/stretch". Illustration from iPad User Guide (`https://help.apple.com/ipad/8/`)

cations; the third party application Loose Leaf[4] also uses a four-finger stretch gesture for copying elements.

However, these novel gestural interaction interfaces are criticized for their poor visibility, discoverability, learnability, and consistency (*Norman and Nielsen*, 2010). As *Norman* (2010) states, "because gestures are unconstrained, they are apt to be performed in an ambiguous or uninterruptable manner", and constructive and continuous feedback are needed "for the user to learn the appropriate manner of performance and to understand what was wrong with their action". There have been many approaches to visual feedback for multi-touch gesture interfaces (for a brief overview see *Buxton* (2007)). Previous works have focused on the ease of recognition and memorization of multi-finger gestures, but mapped arbitrarily to commands (*Ghomi et al.*, 2013; *Wagner et al.*, 2014). Using simulated physical objects for visual affordance has also being explored by *Bragdon et al.* (2010).

In this study, we explored three different visual representations for multi-touch gestures: The menu mode uses only a single "tap" gesture, the icon and gesture modes make use of more complex drag selection or "drag-and-drop" gestures, and both use different methods to visually guide the interactions.

4.2.3 Mobile Multi-Touch Programming

Commercial applications exist for traditional text-based programming or scripting on mobile touch screen devices, most of them developed for the iPad. This is likely due to its large screen size, which allows a full-sized virtual keyboard to be used. Applications such as Pythonista or Diet Coda (*Zorn*, 2012; *Panic*, 2012) are examples of direct translations of desktop-based code editors for multi-touch. More experimental editors such as Codea and ScriptKit (*Saëns*, 2011; *Buza*, 2012) augment the basic text editor with templates that the user can drag and drop into the code by touch, and in-place visual editors to automate programming tasks such as picking color or texture; text-based scripting languages such as

[4] `https://getlooseleaf.com`

Lua are still used. Beyond tablets, mobile Integrated Development Environments (IDE), such as AIDE [5], also support text-based programming on phones with smaller screens as well as tablets. Mobile programming has also been used to reinvent programming as a social gaming experience (*Berland et al.*, 2011), highlighting the benefits of programming on the move and while standing.

The urMus environment (*Essl*, 2010b) is designed with Musicians in mind, and features both a web-based text editor for programming using the Lua scripting language and a "drag-and-drop" programming interface. Environments such as Hopscotch (*Leavitt et al.*, 2013) and Catroid (*Slany*, 2012) adapt a visual programming interface, where draggable blocks that can be sequentially stacked represented code statements. AppInventor (*Wolber et al.*, 2011) allows the user to program for mobile devices through a web browser, using a block-like visual programming interface. LiveCode (*Holgate*, 2012) uses a natural-language-like scripting language for programming mobile devices, but still requires a PC to do the actual coding. More recent environments such as TouchDevelop (*Tillmann et al.*, 2011) adapt a similar menu structure interface, where tapping on evolving menu entries assembles pre-built keyword blocks. A field study of mobile programming on TouchDevelop platform was conducted to examine the usage patterns in real users (*Li et al.*, 2013). *McDirmid* (2011) presented a similar mixed menu-iconic approach for touch-centric interaction paradigm for textual programming on tablets.

Many game-like visual sandbox environments also contain elements of visual programming. Examples include Creatorverse (*Linden Research, Inc.*, 2012) and Blocksworld (*Linden Research, Inc.*, 2013), where a 2D or 3D physics engine allows users to build mechanical constructs using "drag-and-drop" gestures and apply some constrained vocabulary of interactivity or logic to these constructs.

Pong Designer (*Mayer and Kuncak*, 2013), which combines a 2D physics engine with directly manipulatable objects in a sandbox, is closest to our approach in general mobile

[5] AIDE: http://www.android-ide.com

programming. It enables users to add programming logic by inferring event and causal relations to build simple games (e.g. when a ball collides with a wall, increment a counter).

The common graphical representation of our system uses a visual-sandbox like environment (similar to Blocksworld) for assembling elements, eschewing text-based coding. Unlike most environments where a program is assembled and then ran, our system is fully interactive and there is no distinction between modes of assembling a program versus running one. In the design of the different interactions and representations modes, elements of the mentioned visual sandbox environments are incorporated such as "drag-and-drop", and multi-touch gestures such as "pinch". Our focus is not to build a full-fledged programming environment but to compare different visual representation and interaction modes using programming-like tasks. The constructive capability is limited in scope to building dynamic interfaces (e.g., complex branching behavior in procedure languages is not supported in our system, currently).

4.2.4 Visualization for Gestures

Earliest examples of gesture-capable interfaces such as the marking menu (*Kurtenbach et al.*, 1993) predate the popularity of touch-screens. Existing work has found visual feedback to be beneficial in mouse and pen-based direct manipulation interfaces (*Wilcox et al.*, 1997; *Shneiderman*, 1983). In pen-based interactions, visual information is also found to be more effective than sound as a feedback mechanism, due to better visual-spatial memory in most users (*Zhai et al.*, 2012). On touch-screens, visual feedback is also found to have a strong effect on accuracy in pointing/crossing tasks (*Luo and Vogel*, 2014).

One of the central criticisms by Donald Norman on gestural interfaces used in consumer electronics is the lack of feedback to guide learning, as well as execution, of gesture commands (*Norman*, 2010). Previous works have addressed this by using simulated physical objects for learning (*Bragdon et al.*, 2010). Continuous and dynamic visual guides, which aim to suggest ways to complete a gesture in progress, were also proposed (*Vermeulen et al.*,

2013). Multiple works used continuous visualization to show possible commands given the current state of interaction (*Bau and Mackay*, 2008; *Lundgren and Hjulström*, 2011; *Ghomi et al.*, 2013; *Freeman et al.*, 2009; *Sodhi et al.*, 2012), usually showing potential gesture choices located either right at the site of the interaction (near the finger touch location), or mirrored at a more visible location.

In icon and gesture interaction modes in our system, dynamic continuous visual feedback is used to suggest possible outcomes associated with gesture commands, but it is not the primary focus of the study.

4.2.5 Impact of Device Size

The impact of mobile device size has been the subject of investigation from a number of perspectives. In particular, the impact of screen size has been studied with respect to subjective usability and efficiency measures on different sized mobile phones (*Raptis et al.*, 2013), with larger size mobile phones found to be more efficient. Similarly, psychological factors such as enjoyment, perceived mobility, and intention to use were studied across mobile phone and tablet form factors. While mobility decreased as device size increase, efficiency and usability measures were lowest for intermediate sized devices and higher for small and large form factors. For basic pointing tasks, large touch screens were also found to be preferable to smaller ones (*Oehl et al.*, 2007). Larger touch screens were also reported by users to lead to higher enjoyment than smaller ones (*Kim et al.*, 2011). Our work is unique in examining the effect of touch screen size on programming-like tasks using a variety of interaction and representation modes.

4.3 Representation & Interaction Design

A prototype of the programming interface is built using Lua scripting language on the urMus (*Essl and Müller*, 2010) platform. urMus provides a set of Lua API for programming mobile phone sensors, audio and graphics, with which a block-based drag-and-drop interface can be

easily implemented. For our purpose, urMus serves as a flexible general-purpose platform for prototyping representations and interactions. The implementation detail of our system is described in **Chapter VI**.

4.3.1 Shared Grammar

The basic grammar of the visual environment starts with a *full-screen* canvas where the user can create basic elements called *regions* and arrange them spatially as well as other elements that are discussed below. This is where a user creates a dynamic interface (see Figure 4.14 for example). Next, we present the general programming and interaction elements that will be shared between all representations.

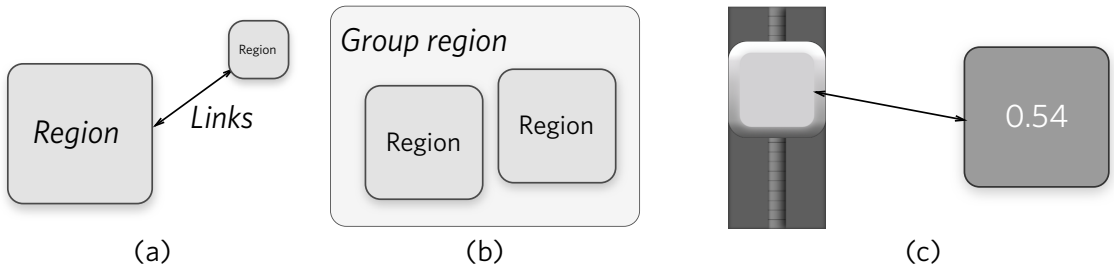

Figure 4.3: Basic elements of our environment include (a) Regions and links (b) groups for organization and spatial constraints. A simple example interface which can be easily built using these primitives is a slider in (c)

4.3.1.1 Region

The most basic building block is a *Region* (see Figure 4.3 (a)). Each region is visually represented by a rectangle on screen and can be directly manipulated via dragging and resized via a pinching gesture. Regions can be created with a simple tap gesture on the canvas, and can be arbitrarily arranged on the canvas. Their visual appearance can also be modified, and they can be pinned to the canvas to prevent movement, which can be useful if it is used as a fixed interface element (a button for example).

Regions possess characteristics of a generic variable or object in the more traditional sense, in that they can be used both as abstract containers for values, such as a variable or a

constant, and as a visual object in the program. All regions can send and receive events, in a manner similar to function calls. Regions can be configured to have other functionalities as well, such as playing an audio sample (which may be useful for building musical instruments) or moving its position on screen. Similar to how class can be instantiated as many times as needed, regions can be duplicated while retaining their properties as well as links which define how they interact with other regions. In all representation modes, tapping on the empty canvas creates regions and their appearance is changed using a common texture picker interface (see Figure 4.4).

4.3.1.2 Links

Following the concepts of node and edges in event-driven data-flow programming, each region can receive *events* and respond to them with *actions*. The routing of these events between regions constitutes the main method of building interactions in the visual environment. Links are visually shown as lines directing from one region to another, similar to visual patching interfaces. Each link is directional and stores an event type that is generated from the sender of the link and an action that is to be taken by the receiver when the event is detected. In the example in Figure 4.3 (c), a region on the left is linked to send its vertical position value to the region on the right (which displays a numerical value), acting as a vertical slider.

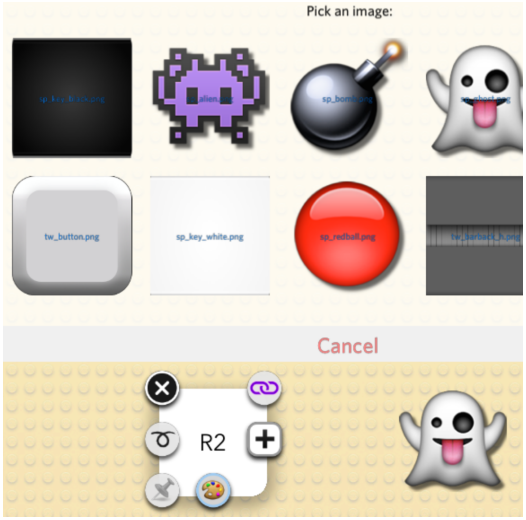

Figure 4.4: The image picker interface for changing a region's texture. This is common for all three representation modes.

This mechanism can be used for building more complex interactions, since a region can respond to multiple events from multiple sources, as well as respond to each event with multiple actions, or forward events to other regions. Events can include touch-based

events from user input (e.g., fired when the region is dragged, tapped, or held etc.). The actions that can be triggered by events can change the properties of the region (e.g., size, position or movement), or can be forwarded to another region. When a region is duplicated, all the incoming and outgoing links are also duplicated, preserving its interaction between other regions.

For simplicity of the experiment, we implemented a basic set of touch-based events and actions. The move event is evoked when the region is dragged; this sends the current position of the region, which can be responded by a move action, which moves the receiving region in the same direction and distance. The receiving region can also respond by displaying one of the coordinates of the received position and sending it to a music synthesizer.

4.3.1.3 Groups

We use groups to encapsulate and organize sets of regions. Similar to how objects can be nested in object-oriented language, group regions function both as conceptual containers for regions and as a way to spatially organize objects. Because we treat groups as container regions (implemented as a subclass of the region class), they can also receive and respond to events. When a group is duplicated, all its children are also duplicated. Visually, a group region spatially constrains the movement of its child regions, so they cannot be moved out of the group. It can be used to create common interface widgets such as sliders (see Figure 4.3 and Figure 4.14 for examples).

4.3.2 Visual Representation and Interaction Modes

Given the shared mechanisms described above, we then implemented three types of representations for manipulating and interacting with them: (1) Menu-driven, (2) Icon-driven, and (3) Gesture-driven.

4.3.2.1 Menu-Driven Mode

As a baseline for comparison, we created a menu representation mode that has more similarity to a traditional desktop UI. Except for region deletions, link deletions and the shared gestures mentioned above, all commands such as creation of links or groups are activated through a contextual text-based menu (see Figure 4.5) that is associated with one region. Deletion buttons for each region and link are shown whenever a single tap activates the menu.

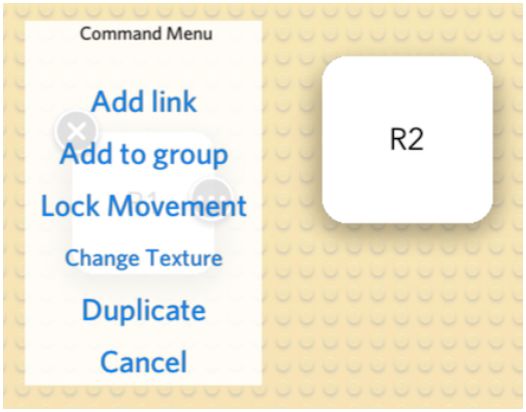

Figure 4.5: The text-based menu associated with a region. Deletion command is accessed through the top left button instead of the menu.

For commands that require a second region to act on (e.g., creating a link between two regions), the user activates the command and then is prompted to select the second region (e.g., the target region to create a link to, or in the case of creating a group, the region to be used as the container) by tapping. After the selection, the command is executed on the two regions. Figure 4.6 shows an example of creating a link using the menu system, as well as the menus for selecting events and actions. The creation of groups uses the identical steps.

Due to its widespread use on different platforms, we expect this representation/interaction mode to be most familiar to average users who have prior experience with desktop PCs and commercial touch screen operating systems such as iOS or Android, where text-based menus are common.

4.3.2.2 Icon-Driven Mode

In icon-driven mode (icon for short), most commands are accessible through a contextual symbolic menu surrounding the region (see Figure 4.7). The icons are arranged similar to a radial layout surrounding the region, which borrows from previous work on radial menu

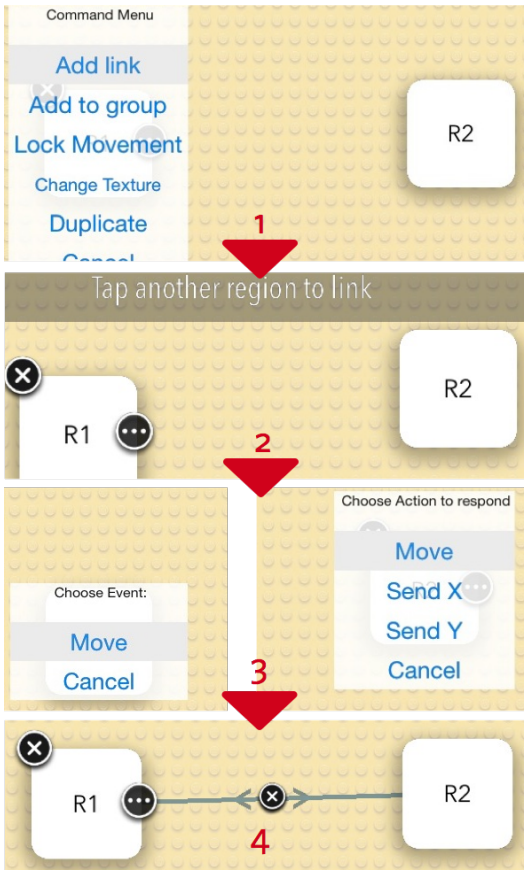

Figure 4.6: Steps for creating a link using text menu mode. 1) The link creation command is selected from the menu, 2) the user is prompted to select a second target region by tapping, 3) events and actions are selected using their respective contextual menu, 4) at last the link is created. Creation of groups uses the same mechanism, minus step 3 since no events or actions is needed.

design for touch-screens (*Kammer et al.* (2010) for example).

Two types of gesture interactions exist in the icon-driven menu. Static buttons are activated by tapping (e.g., the "Delete" button on the top left corner), while draggable buttons function as handles for more complex "drag-and-drop" gestures. These drag gestures activate different types of semantically related commands. For linking, the link icon can be dragged and dropped on the region to link to, and the potential link is visually shown using a cable-like metaphor through the interaction (see Figure 4.7). The grouping gesture handle is used as a lasso selector to select other regions to add to the parent group region (see Figure 4.8). The copy handle, visually represented as a smaller version of the region,

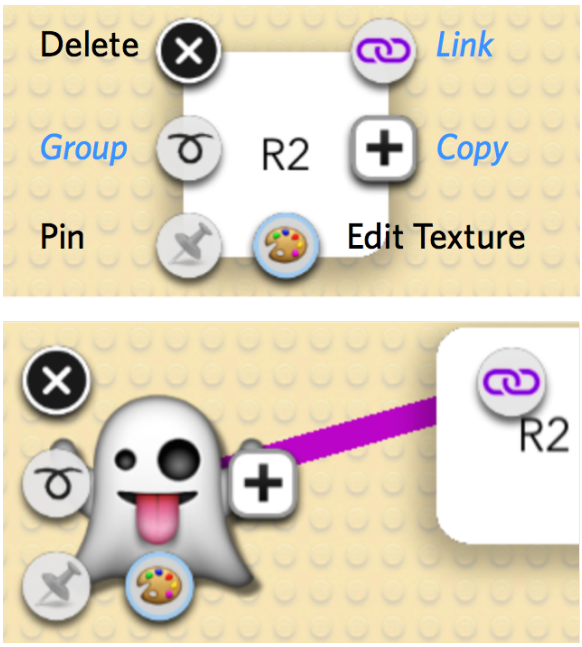

Figure 4.7: Above: Icon mode contextual menu, actual interface does not contain the text labels. Pin button anchors the region on canvas, edit texture is used for changing the appearance of regions. "Group", "Link" and "Copy" icons, which are labeled in blue, are draggable gesture handles, they are periodically animated for distinction. Below: an example of the draggable icon handle for linking.

can be tapped to produce a copy of the parent region. It can also be dragged into any other area on screen to produce a copy of the parent region at that location when released (see Figure 4.7).

To visually differentiate draggable gesture handles from the static buttons, the draggable handles are animated periodically when not used, moving slightly away and back to their original position, as visual affordance suggesting that they can be dragged as opposed to responding only to taps.

4.3.2.3 Gesture-Driven Mode

The last representation mode is designed to be almost entirely driven by direct manipulation gestures that act on the regions. In our approach, gestures are designed for directly manipulating the on screen elements (regions in this cases). The mapping between gestures and the associated command is not arbitrary, but is semantically related to and reinforces

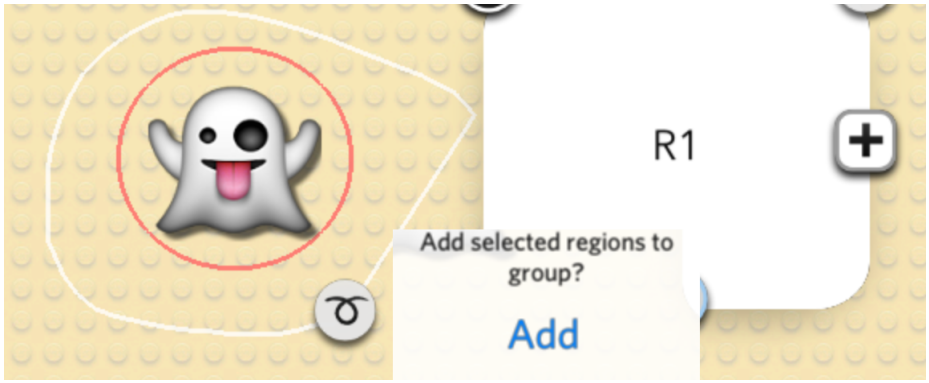

Figure 4.8: Draggable handle used for selection of regions to add to a group in icon representation mode

the command being triggered.

For linking and unlinking two regions, a pinch or stretch gesture is used (see Figure 4.9). A pinch gesture, where two regions are dragged concurrently and moved towards each other, is used for linking. The opposite, moving away from each other, is used for unlinking, reinforcing the concept of linking and unlinking. For each gesture, the regions have to be moved past a threshold (which is visually shown when reached) for the action to be completed. This acts as a confirmation for each action to prevent mistaken activations. If the threshold is not crossed, regions are automatically restored to their previous positions and the action is cancelled.

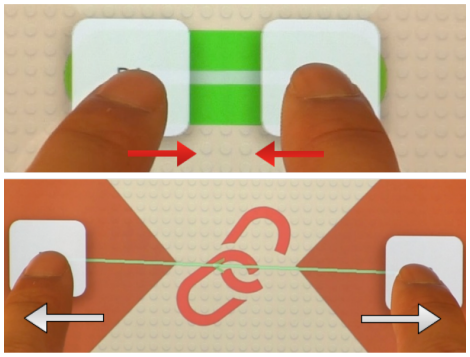

Figure 4.9: Gesture mode uses pinch gestures to create (above) or delete (below) links. Background color changes show the current command that will be executed when the gesture is ended by releasing the hold, as a guide through the activation of the pinch gesture.

A background color guide is displayed faintly after the initiation of the gesture as a visual guide, and is continuously updated depending on the user's progress in completing each gesture (i.e., "pinch" or "stretch"). The color area corresponding to the potential command is displayed in increased intensity, while the color corresponding to the opposite

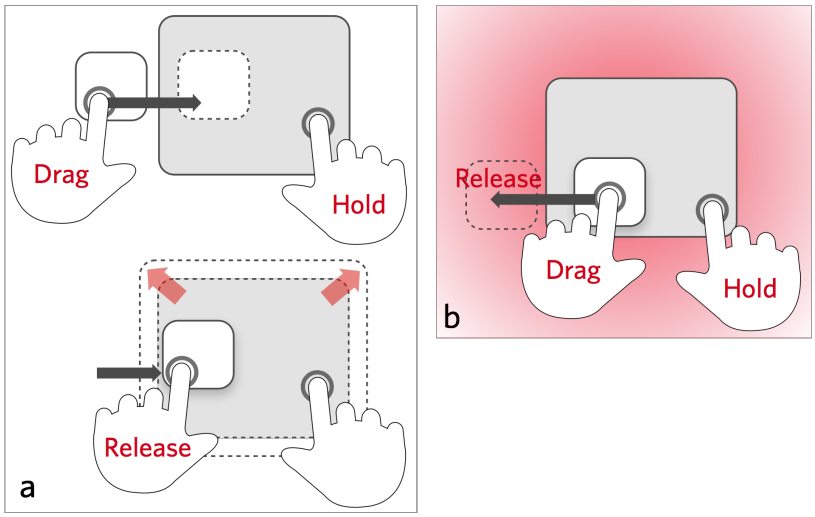

Figure 4.10: A drag-and-drop gesture is used to add a region to a group (a), the reverse removes a region from a group (b). In both cases, visualization shows the progression of the gesture command, in (a) the potential group region is temporarily enlarged; in (b) red background show a drop-zone for the impending drag-and-drop gesture.

command is faded (see Figure 4.9 for the visual guide). The large color background is meant to alleviate the problem of occlusion by the user's hand.

For grouping regions, a drag-and-drop gesture is used. When two regions are dragged, and one is released over another one with a larger size, the smaller or released region is added to the group associated with the larger region (see Figure 4.10). To remove a region from a group, the reverse gesture is used: the user simply drags both the region and its parent group region, and then moves the child region outside of the group region and releases. The movement restriction of child regions within their group region is temporarily lifted while this gesture is performed, and restored after the remove-from-group action is either executed or cancelled. Similar to the linking gesture, visualization provides guidance by enlarging the potential group region when adding, or showing a drop-zone when removing.

The region creation gesture of a single tap on the empty canvas (shared among all three interaction modes) is modified to support copying. While holding onto a source region to be copied, tapping the empty space on the canvas creates instead a copy of the source region (see Figure 4.11). Since there is no need to release the hold on the source region,

this allows for efficient creation of copies.

To pin a region, a double tap gesture is used to toggle between restricting and allowing movements (see Figure 4.12 (c)). For deletion or modifying a region's texture, a hold-and-slide gesture menu is used. The gesture menu is displayed after holding onto the region for a short time and while no other commands (e.g., moving, resizing etc.) are activated. The two commands are activated by maintaining the hold gesture while sliding towards the command icon (similar symbol as the icon mode) and then releasing the finger. Figure 4.12(b) shows the visual feedback for activating each command after the hold-and-slide gesture.

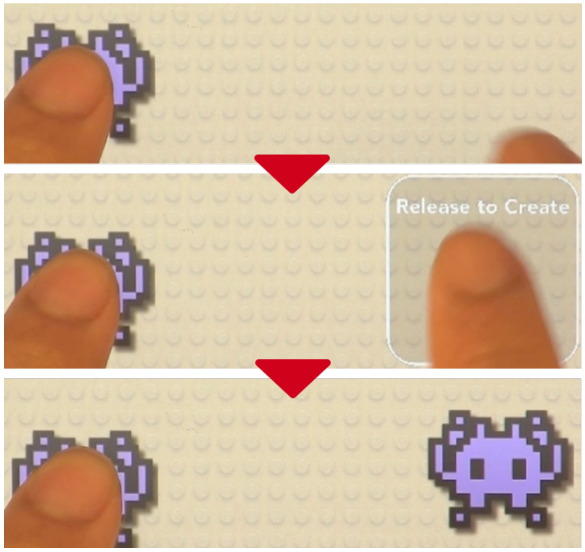

Figure 4.11: Copying regions in gesture mode uses a modified tap-to-create gesture normally used for creating new regions. Holding on a source region while tapping the empty canvas to create copies.

4.3.3 Considerations in Touch-Based Interaction

For all three interaction modes, touch targets, if present (i.e., each item in the text menu, or the icon based buttons in icon mode), are designed to have a minimal size of 9mm (on the higher pixel density 7.9" iPad Mini) as recommended by standard mobile user interface guidelines (*Apple Inc.*, 2013; *Google Inc.*, 2013).

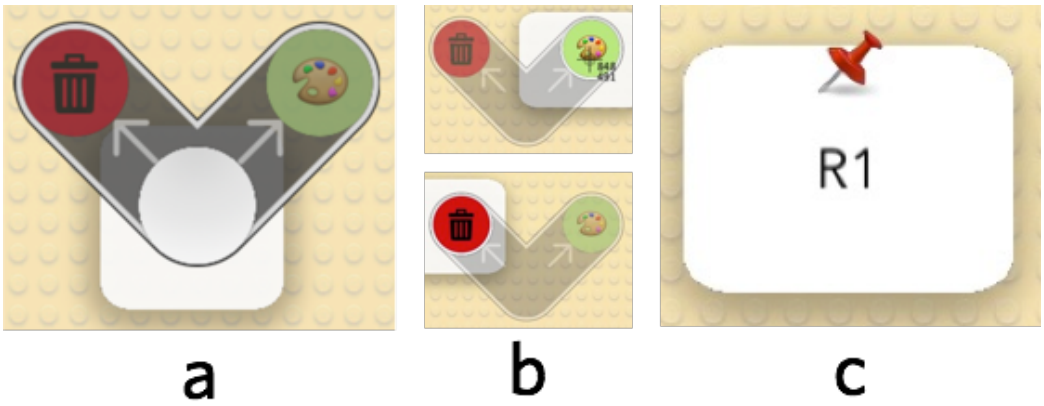

Figure 4.12: (a, b) hold gesture activated menu and its different states, (b) shows how the non-active commands is faded out while the active one is increased in opacity. (c) Visual feedback for when a region is pinned after double tapping

Continuous visual feedback is also used in interaction modes that rely on drag gestures (i.e., drag-and-drop, pinch, lasso selection etc.). In previous work continuous visual feedback has been found to be beneficial in visual programming contexts (*Wilcox et al.*, 1997). In icon mode, the cable and lasso selection visualization are continuously animated, and in gesture mode the color background is continuously updated during the linking gesture (see Figure 4.9).

4.4 Experiment

We conducted a task-based user study to explore the efficacy of different interaction and representation modes and device sizes, using the shared visual programming environment. In particular we are looking to test a set of three hypotheses: **H1:** Does the mode of interaction significantly influence performance? **H2:** Does prior experience affect how effective or different modes are? **H3:** Does larger device size lead to better performance? The user study is designed to test these hypotheses. Further, we collect subjective evidence via questionnaires as well as analysis of details of interaction patterns across all three modes.

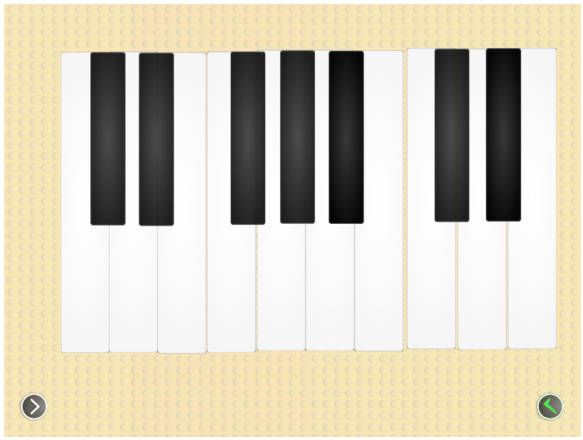

Figure 4.13: A touch screen piano keyboard interface built with our environment. Participants were asked to build this in Task 1.

4.4.1 Design

We used a mixed within-subject factorial design. The factors included mode of interaction (menu, icon and gesture, as mentioned before) and device size (full sized iPad with 9.7" screen and iPad mini with 7.9" screen).

All three modes of interaction were presented to all participants (within-subject), while device sizes were randomly and uniformly distributed among participants. Each participant is assigned to only one of the two device sizes. The presentation order of interaction modes was randomly assigned and counter-balanced. This, in turn, is in anticipation of possible learning effects in the data and allows inclusion of presentation order in the statistical analysis.

As described before, for the mode of interaction, the underlying semantics and elements of the visual environment, such as Regions and Links are held constant, while the interaction mode for manipulating these programming primitives is varied.

For the different device sizes, the identical interface is scaled to each screen since both have the same aspect ratio and the same effective resolution of 1024x768 pixels. This means that fixed sized interface elements, such as menu and icon buttons, are 38% larger on the full size iPad.

We designed two tasks for the study. Task 1 asks participants to build a touch screen version of a simple piano keyboard interface (see Figure 4.13). This task asks participants to arrange regions in a layout and change textures. Since the keyboard contains multiple regions sharing the same texture, participants are encouraged to use the duplication function extensively. Task 2 asks participants to build a music controller interface with sliders and a 2-D control pad, making more use of grouping and linking functions (see Figure 4.14, simplified to contain only two sliders). There are many possibilities for programming-like tasks, but our selection is not meant to be representative, only to ensure coverage of most of the basic possible actions. For this reason we did not consider task as a factor in our analysis.

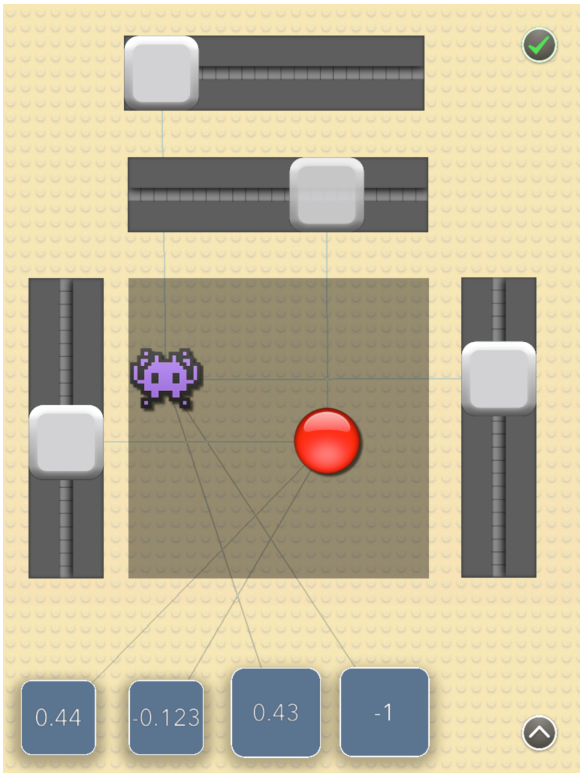

Figure 4.14: An example of interface built with our environment: a music mixer control panel containing multiple sliders as well as a 2-D pad for simultaneous control of multiple parameters. The sliders widgets are linked with the 2-D pad so they move concurrently whenever the other one is used to adjust values. The output parameters are fed into grey inlet regions at the bottom that can fed the values to a music synthesizer. Participants are asked to build a simplified version of this in Task 2 with only two sliders.

Each participant is randomly assigned one type of device out of the two sizes. After a short introduction to the basic grammar of the programming interface, each interaction mode is introduced. Five minutes is allotted for learning on each interaction mode to help mitigate learning effects. During each learning period, the participants are asked to perform basic actions (i.e., creating regions, changing visual appearance of regions, creating links etc.) until they are able to complete the actions without guidance.

After each learning period, Task 1 is presented. After participants complete the task three times, one on each interaction mode, they are asked to give qualitative feedback on each of the interaction modes through a survey. The survey evaluates each interaction mode in terms of usability, discoverability and difficulty to learn. The survey contains 13 Likert scale questions, and takes less than five minutes to complete.

After completing the survey, Task 2 is presented, and the participants are asked to complete it three times using the same interaction modes as Task 1 and in the same order. Afterward, a nearly identical survey is given regarding Task 2, and regarding their general impression of each interaction mode. In total, the six trials (three on each task) last approximately 70 minutes.

Using the underlying urMus framework, we instrumented each interaction mode so that every interaction participants have with the device is recorded. This includes low level touch events (i.e., touch down, up, drag events) as well as the activation of each command that modifies regions, links or groups. The recordings are analyzed in aggregate for each task with each mode. The following aggregate measures are considered for each trial:

- **Task Completion Time**: The participants are instructed to complete each task as fast as they can and to tap an onscreen button when they finish. We define completion time as the duration between the first interaction and the last interaction before the *finish* button is tapped. Longer completion time indirectly reflects the difficulty participants encountered while using each mode.

- **Total Wait Time**: From the interaction log we also measure how much time each participant pauses between each interaction sequence (activating a menu, moving or touching a region etc.). The cumulative wait time across each task may indicate user confusion or difficulty in recall.

- **Total Actions Performed**: Beyond the low-level touch events recorded, we want to look at higher-level actions that actually manipulate properties of an element (e.g.

changed texture, locked movement, created links and etc.). Instead of considering each interaction such as tap or drag, we consider actions independent of the sequence of interactions (i.e., whether a menu tap or a gesture is used) used to activate them. This gives us an indirect measure of how efficient each mode can be. When looking at total actions performed to accomplish each task, since the end goal of the task is the same for every trial, more actions to reach the same goal may indicate redundancy or more error correction. We estimate this as well using the number of deletions since there is no undo built-in. When normalized over time, the number of actions can also indicate how efficient each mode is at supporting fast activation of actions, although not necessarily how useful.

- **Drag Time**: We aggregate drag events that are generated at display refresh rate every time a finger is dragged on screen into more discrete drag "strokes" (see Figure 4.16), where each drag gesture starts with a finger touch-down event, continues with a series of drag events, and ends with a touch-up event when the finger is lifted from the screen. We measure the total time spent on drag gestures as well as the average duration of each drag gesture. Since the different interaction mode relies on drag-like gestures to different degrees, we hope to see if this affects the kind of drag gestures users perform, as well as the overall accuracy required for these gestures. Assuming Fitts's Law can model the drag gestures, the drag gestures requiring high precision will necessitate a slower, longer drag.

4.4.2 Recruitment

We recruited 21 graduate and undergraduate students from the University of Michigan. We selected participants from a variety of academic concentrations with an average age of 26.7 years, 40% of which are female. Most participants have a significant amount of experience using touch screen phones (mean = 5.29 years, std = 2.17), about 40% have extensive experience using tablets (mean = 2.35, std = 2.00), and 33% having more than 5 years of

programming experience, while 28% with less than two years (mean = 4.76, std = 4.16). We hope to capture a broad spectrum of experience in both programming and touch screen usage.

4.4.3 Results

Since the randomized distribution in experiment design did not result in an equal number of participants in each condition (i.e., presentation order of modes, device size), direct ANOVA analysis cannot be applied. We used general linear model (GLM) for analysis and we also examined two-way interactions using GLM. Presentation order is included as a factor in the GLM regression to account for the effects of time. We also removed outlier data points outside 2-times-standard-deviation range for each participant, with about 1.4% of data points removed. We use the standard alpha value of 0.05 as threshold for significance. Interaction mode factor has more than two levels, so post hoc pairwise comparisons are done with Tukey HSD multiple comparisons correction (to have alpha value of 0.05 as significant) to examine differences between the three interaction modes.

For many of the metrics we examined below, standard deviations are large between participants within the same conditions, possibly resulting from the complexity of the tasks and the differences in how people approach the tasks.

4.4.3.1 Task Completion Time

GLM shows that device is a significant factor ($F = 5.86, p < 0.0169$), while mode is not ($F = 0.65, p < 0.52$). Surprisingly, the larger sized iPad shows longer completion time than the iPad mini, disproving hypothesis H3. Presentation order is found to have a significant effect on most of the responses ($F = 18.19, p < 0.0001$). Also surprisingly, prior experience with touch screen phone and tablet are not found to be correlated with completion time.

Prior experience in programming, however, is positively correlated with faster performance ($F = 4.76, p < 0.0312$) in GLM analysis. One possible explanation is that prior programming experience and understanding also aid the user in a visual paradigm.

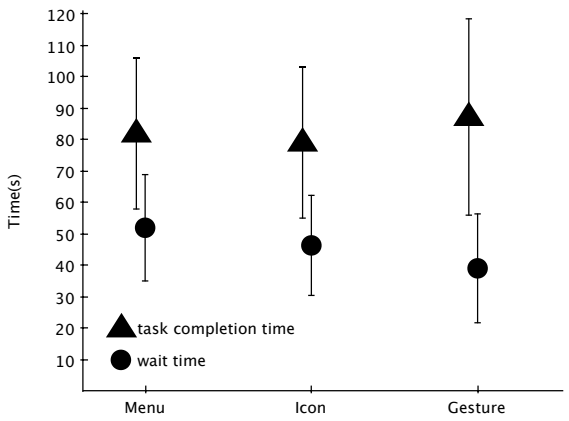

Figure 4.15: Task Completion Time and Wait Time both show high variance between participants within the same condition, however, the difference between gesture and the two other modes are significant.

Given the high complexity of the task and the flexibility, difference due to mode is likely overshadowed by the high variance in individual participants, and task completion time is possibly a poor measure of complex programming tasks (See Figure 4.15).

4.4.3.2 Wait Time

For wait time, mode and device are significant factors ($F = 4.30, p < 0.0156$,$F = 8.08, p < 0.0052$, respectively). The order of presentation is significant ($F = 10.55, p < 0.0001$). For mode, post hoc comparisons show menu ($mean = 51.99, stdev = 16.91$) and icon ($mean = 46.39, stdev = 15.94$) lead to significantly more wait time than gesture ($mean = 39.10, stdev = 17.28$), menu and icon are not significantly different. This is likely explained by the extra effort required to read menu command text or icon symbols, whereas in gesture there are very few symbolic or textual visual cues. At the same time, as shown later with drag time analysis, gesture mode may simply require longer, extended time to perform each action and so it leaves less time for pause. For expertise, only experience in tablet usage is significant ($F = 6.80, p < 0.0102$).

4.4.3.3 Total Actions

For total actions performed, we found mode to be a significant factor ($F = 18.98, p < 0.0001$), with gesture having significantly more actions than both menu and icon ($p < 0.0001$ from post hoc comparisons, icon and menu are not significantly different). (As expected, task is significant ($F = 84.37, p < 0.0001$) due to the difference in task design.) Device is also significant ($F = 14.03, p < 0.0034$), with the larger iPad having more actions performed. Presentation order is not significant ($F = 0.64, p < 0.5876$). Experience in tablet usage is possibly correlated ($F = 2.91, p < 0.0583$) to total actions, as participants with more experience use fewer actions to complete the tasks.

When examined in combination of percentage of delete actions, we can infer how often the user is correcting a mistake in creation of links, regions or groups. For percentage of deletion out of total number of actions, mode is significant ($F = 3.08, p < 0.0499$), in particular menu has higher error correction rate than gesture (mean = 9.644% vs mean = 7.775%, again this significance does not show in pairwise t-test comparisons), suggesting that perhaps participants create more regions, links and groups by mistake with menu than with gesture. Icon mode has similar deletion rates as menu mode at mean = 9.23%. The fact that gesture tend to have more actions in total but a lower percentage of deletions, suggests that the extra actions, which are not deletion, have more to do with refinement (e.g., changing appearance or size).

When we considered the rate of action (normalized over time), GLM regression shows that mode is also significant ($F = 14.18, p < 0.0001$), with gesture mode having significantly higher rates of action than icon and menu (with no significant difference between icon and menu).

4.4.3.4 Drag Gesture Times

We aggregated how people use drag gestures. When considering the total time used for dragging gesture, we found that mode, task and device are significant factors ($F = 12.01, p <$

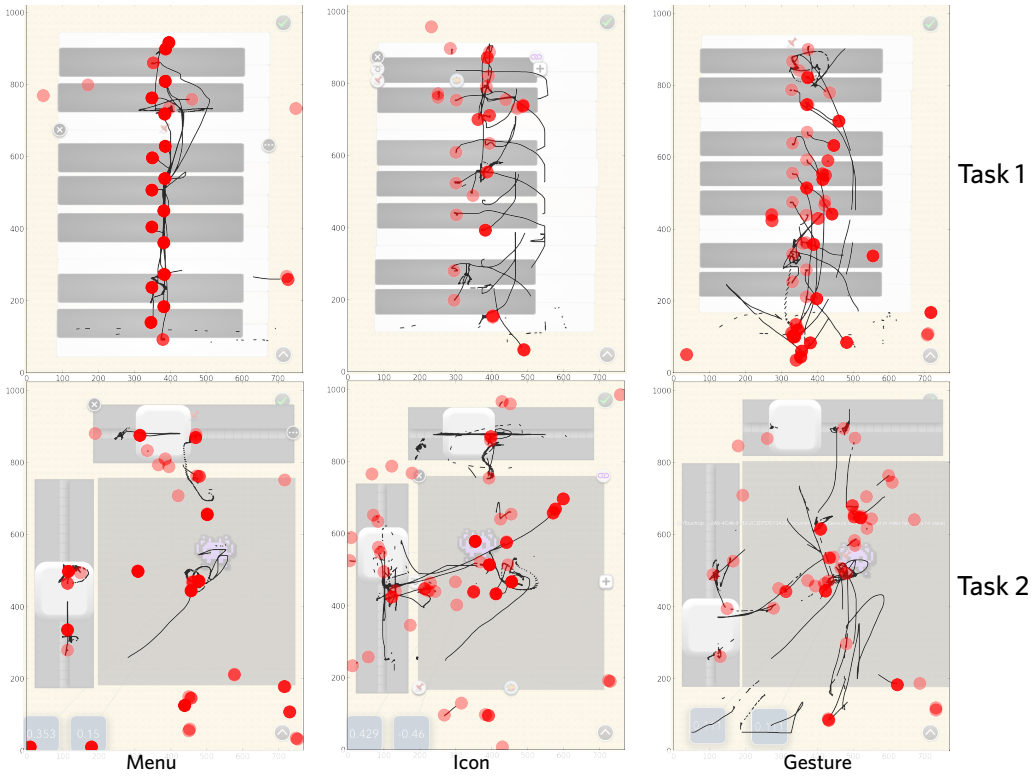

Figure 4.16: Sample trace of all drag gesture and activation of commands for one participant. Columns are interaction modes, and red points denote where each command is activated, regardless of the mode.

$0.0001, F = 6.33, p < 0.0131$, respectively). Post hoc comparison shows that under gesture mode participants spent more time dragging than menu or icon, with no significant difference between icon and menu, which is not surprising given the emphasis of gesture mode on drag-and-drop based interactions.

With device as the factor, participants spent more time on dragging while using the larger iPad. Since the same goals are given for the tasks on both devices, participants naturally tend to scale up the layout with a larger screen, possibly contributing to the longer time. Both devices have identical pixel resolution, and so all onscreen elements with fixed or default sizes such as a newly created region, buttons and menus, are already scaled up on the larger screen.

When examining average duration of each drag gesture, only device is a significant factor ($F = 5.06, p < 0.0264$), as the larger iPad again shows longer average drag duration

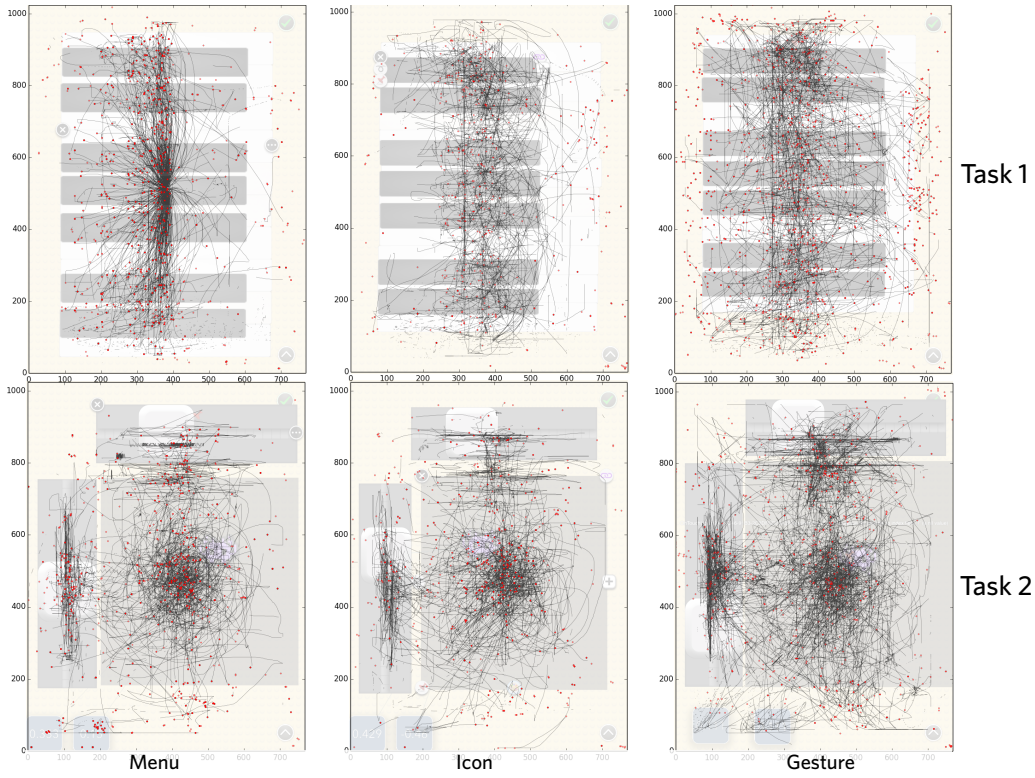

Figure 4.17: Traces of all drag gesture and activation of commands across all participants

than small iPad. With respect to average drag distances, neither device, task nor mode is a significant factor.

We visually examined the aggregated drag gesture traces recorded from each trial. Representative samples can be seen in Figure 4.16, and aggregation of all participants' drag gestures can be seen in Figure 4.17. The trace shows fewer gestures in menu mode while more in gesture mode. This was not surprising given the design of each mode. It also shows more short stop-and-go drag gestures in gesture mode and icon mode, where the participant drags in a sequence of small steps, possibly iteratively refining the drag. Although both icon and gesture show similar amount of drag gestures, it is likely that the difference in drag time is explained by unfamiliarity with gestures without having icon-based gesture handles in icon mode. Further, the need for more precision in gesture mode in the case of linking and grouping, where a certain gesture threshold has to be crossed for each action to complete, may explain the difference in drag time.

4.4.3.5 Time Effects

We did observe learning effects during each session as the participants completed each task multiple times in completion time measurements, but not in total action performed (as seen in Figure 4.18). We accounted for presentation order by including it as a factor in the GLM analysis, and we also examined possible interactions and found that while order is a significant factor in most cases (typically $F > 10, p < 0.001$), no interaction is found between order and other factors.

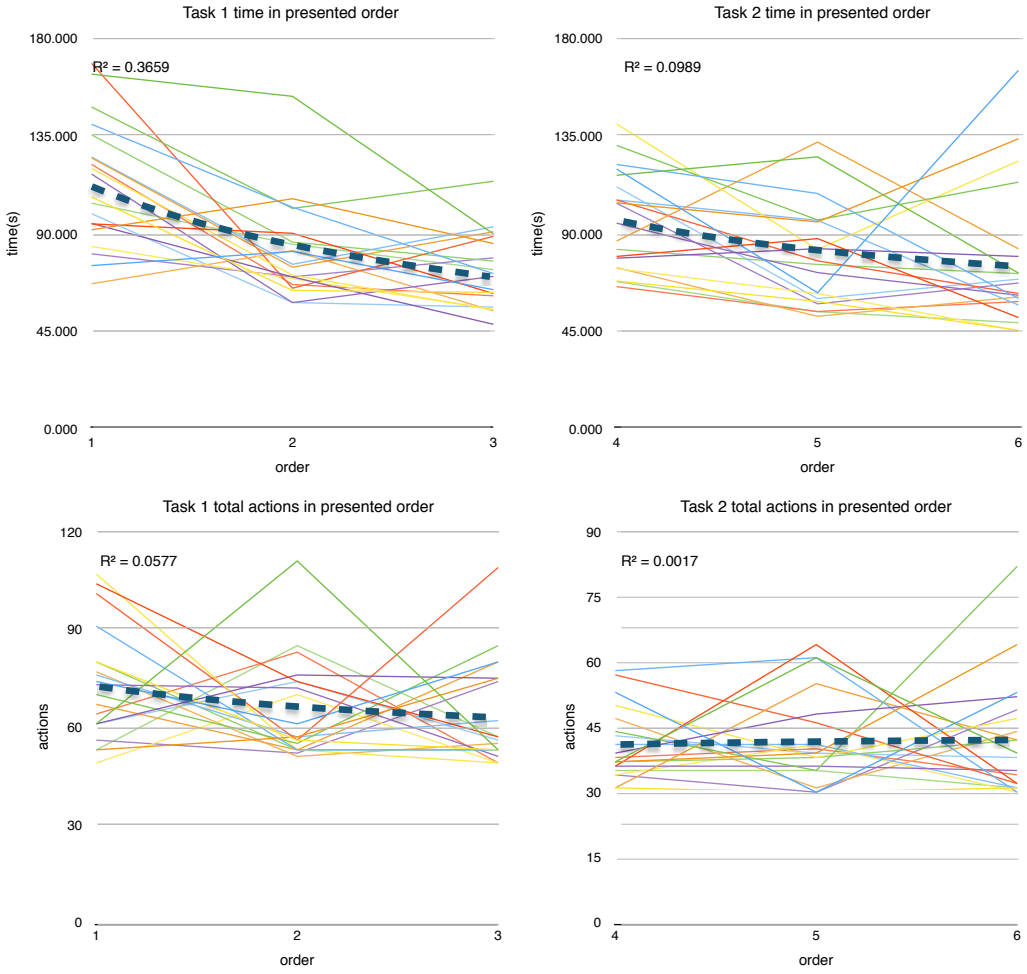

Figure 4.18: Plotting measurements against the presented order of each trial show some possible learning effects in completion time (above), but no obvious trend in total actions (below).

4.4.3.6 Surveys

Thirteen Likert scale survey questions are given after the completion of each task, and we will discuss each of them in term. We included six questions from a cognitive dimension assessment questionnaire (*Green and Petre*, 1996) used in the evaluation of visual programming environments, which are relevant to our system. The Likert scale questions in the questionnaire were analyzed using Mann-Whitney U tests. The interactions between the Likert responses and factors such as device size are examined through General Linear Model where the response values are treated as parametric, with means shown in Figure 4.19.

Questions 1 – 3 assess *ease to learn*, *ease of use*, and *fun to use* respectively. Participants reported that gesture mode is not as easy to learn or use than menu and icon, but found both icon and gesture to be more fun than menu mode ($U = 373$ to $735, p < 0.000002$ to 0.018). Questions 4 – 6 assess how easy it is to manipulate *regions*, *groups*, and *links* respectively. Participants reported that gesture mode is more difficult in all three cases ($U = 344$ to $702, p < 0.00029$ to 0.023) while there's no significant difference between menu and icon. Question 7 assesses how easy it is to make mistakes, and gesture is found to be easier than both icon and menu ($U = 557$ to $601, p < 0.00021$ to 0.00083). This corresponds with earlier observations from deletion actions that show gesture mode usage lends to significantly more actions and a higher percentage of deletions, likely due to mistakes.

Questions 8 – 13 are selected from the cognitive dimensions questionnaire. As our environment is not strictly a system of notation, we selected mostly questions regarding visibility and effects on programming workflow (Table 4.1). We found that gesture mode generally has lower scores than icon and menu, while there is no significant difference between icon and menu modes. Specifically in CD9, mode is not a factor, but visibility is positively correlated with experience in tablet and negatively correlated with programming experience. For CD10, icon mode is found easier than gesture ($U = 735, p < 0.018271$). No significant differences were found in CD12 and 13, although smartphone experience might be positively correlated.

CD8	Is it easy to see the different parts of the "program", and how they function? (visibility)
CD9	Is it easy to make changes? (viscosity)
CD10	Is it easy to stop and check your work so far?
CD11	Is it easy to work in any order you like?
CD12	Is it possible to sketch things out?
CD13	Are any similarities between different parts clear?
CD14	Can you think of ways that the design of the system could be improved?

Table 4.1: List of cognitive dimension questions

The last set of open-ended feedback questions ask participants how each mode can be improved (CD14), as well as any general feedback. Participants generally reported that menu is the easiest to learn, but can be tedious or boring. A few complained about having to rely on reading text to understand. Meanwhile, participants found that icon interface can be hard to understand with only symbols and no text explanation, and that the discoverability of drag-and-drop gestures is low, but fun to operate once they become familiar. There were no reports of fatigue in using any interaction modes.

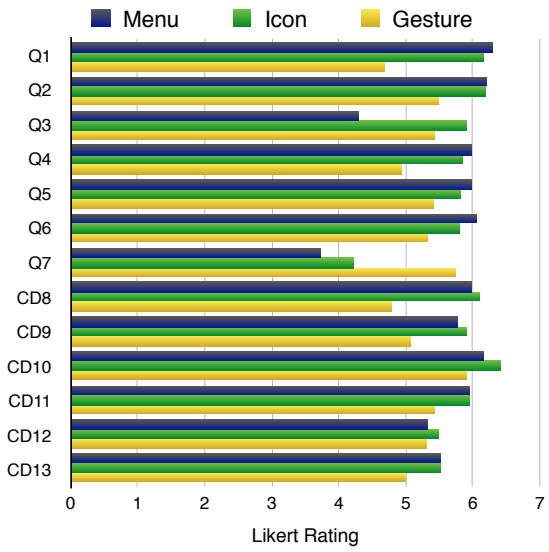

Figure 4.19: Means of all Likert scale responses. Higher are more positive, except for Q7, which asks if it is easier to make mistakes. Shows generally a disadvantage for Gesture mode, except for Q3 where gesture and icon modes are viewed as more fun than menu.

4.5 Conclusion

We presented an experimental study of visual programming representations and interactions based on a shared underlying grammar structure that investigates the impact of device

size, programming background, and user expertise. We found that, contrary to our initial hypothesis, smaller tablets offer faster programming performance across all visual interaction paradigms. This is shown in less task completion time and wait times, as well as less programming actions performed, and suggests a shared underlying explanation.

Out of the three programming representations, the icon-driven mode along with menu mode, consistently out-performed gesture-mode in task-completion time measures and lower wait-times between actions. Gesture mode also has a higher number of drag gestures, which take more time than on other modes (not surprising given the higher reliance on multi-touch drag-gestures), as well as a higher number of total programming actions. Since the goal of each task is identical, the possibly redundant action points to higher numbers of mistakes. In some sense, this disadvantage of gesture-mode is not surprising. Most users had prior experience with menu-based interfaces or icon driven interfaces, which are popular already on existing computational devices, while only 40% of the participants of the study reported any experience with commercial tablets. The short training time for each participants also means each interaction mode, especially the gesture-driven mode, are relatively new to them, in contrast to something like keyboard and mouse with which they have many years of experience. Nevertheless, we do find that prior programming experience correlates to significant higher performance across all representations, suggesting some transferability of programming expertise.

We also evaluated of different modes of representation with subjective questionnaire and a subset of the cognitive dimension model. Subjective feedback shows that there are differences in enjoyment and comprehension between the visual programming modes. In particular, the menu mode and icon mode are similar in performance measures but subjective evaluations found the icon mode to be preferable in terms of enjoyment.

These results also suggest that one should be cautious in making predictions about complex interactive task performances on mobile devices based on the device size. The design of touch gestures needs to take the screen size of the device into account, as we

found that when efficiency can degrade with increased device size. This difference between device sizes is particularly interesting, and in the next chapter we present a study to model basic drag gestures on different device sizes.

CHAPTER V

Fitts's Law and Occlusion on Touch Screen Drag Motions

In the previous study in **Chapter IV** we showed that in a complex visual programming task using touch screen devices, device/screen size is a significant factor in several aggregated performance measures, and that smaller devices lead to better performance. We had previously conjectured that larger screens would lead to better performance since more space is available for complex gestures and assembling operations to be done with more ease. One potential explanation for our surprising finding is that the smaller screen size leads to an overall reduction of distances when performing drag gestures, and the effect of this dominants the advantages, if any, of a larger screen. If this is the case, it should be possible to model drag gestures on touch screen using Fitts's Law (*Fitts*, 1954), which models linear ballistic motion in target acquisition tasks.

Additionally, occlusion by the user's hands and fingers is a common occurrence for touch screen interactions, particularly in the previous scenario when multi-touch required more than one finger to be on the screen. Although Fitts's Law has been adapted to account for varying pointing accuracy of touch screens (*Bi et al.*, 2013), it is not clear that it can directly model dragging gestures with potential hand and finger occlusions as a factor.

Since the previous study was not designed to test for Fitts's Law effects or occlusion under controlled conditions, we conducted a second follow-up experiment using Fitts's Law model to investigate if device size and hand occlusion have a significant effect on perfor-

mance in a touch screen drag task. Specifically, we want to address the following questions: (i) does Fitts's Law adequately model dragging motion on touch-screens, (ii) is performance difference across varying screen sizes (as seen in our previous study) consistently explained by Fitts's Law model, and (iii) what effect does occlusion of hand have on touch-screen drag motions?

5.1 Background

5.1.1 Fitts's Law

Fitts's Law traditionally models a one-dimensional pointing task using hand/arm motions, and is typically defined as:

$$T = a + b \log_2(\frac{D}{W} + 1)$$

with T as time, D as distance (or amplitude) of the movement, W as the width or size of the target, and a and b are constants dependent on the particular interface and user. More recent interpretations include Fitts's index of difficulty formulated by quantifying motor skills using information theory (*MacKenzie*, 1992). Previous work sought to expand to two-dimensional tasks (*MacKenzie and Buxton*, 1992) and compare pointing tasks in physical space and virtual space on-screen (*Graham and MacKenzie*, 1996). They found that virtual space pointing behaves differently when participants experienced more difficulty "homing" on the target at the final stage of the task. Some also adapted Fitts's Law for touch-screens taking account of the accuracy of finger touch (*Bi et al.*, 2013; *Jota et al.*, 2013).

Previous works have compared Fitts' Law under different interaction modes such as mouse versus touch screen (*MacKenzie et al.*, 1991; *Forlines et al.*, 2007). These studies also found a difference between dragging and pointing motions, where users exert more effort for dragging motions, resulting in lower performance.

Notably, *Soukoreff and MacKenzie* (2004) formulated best practices for applying Fitts's Law model in 2-dimensional pointing tasks, which takes into account variations in user

performance (also standardized in ISO (2002)). These recommendations guided our analysis below:

1. Shannon formulation of the *Index of Difficulty (ID)* should be used:

$$\text{ID} = \log_2(\frac{D}{W} + 1)$$

   so that *Movement Time (MT)*, the time for each dragging task, is modeled by Fitts's Law as: $\text{MT} = a + b\,\text{ID}$, where a and b are condition specific constants.

2. *Movement Time* (MT) should be used as the primary measure of performance. In a discrete task where consecutive trials are separated by a break, reaction time should be discounted (the time between the start of the task and when the user actually started moving).

3. End-points and error rates of each movement should be collected. End-point scatter data can be used to perform adjustment for accuracy. Specifically, the target width parameter is adjusted based on real performance of the users and the conditions tested, with an adjusted *effective width* $\text{We} = 4.133\sigma$, where σ is the standard deviation of the end-point position along the movement direction within the specific task condition (distance, width, and etc.). With this adjustment, the *Effective Index of Difficulty (IDe)* becomes

$$\text{IDe} = \log_2(\frac{D}{\text{We}} + 1).$$

   The interpretation is that *ID* values represent the movement tasks that users are asked to perform, while the adjusted *IDe* corresponds to the actual movements that the user performed. According to *Soukoreff and MacKenzie* (2004) there are two reasons for possible discrepancy: (i) The spread of movement end-points will not perfectly align with the target width specified, and (ii) users tend to cheat on easier *ID* conditions by not moving fast enough or covering the whole distance.

4. Linear regression should be used to find the parameters a, b of the Fitts's Law equation: MT $= a + b$ IDe and the resulting intercept (a) should be close to zero (under 400ms in most previous work). A large intercept might indicate problems with the methodology.

5. *Throughput (TP)* (TP $= \frac{\text{ID}}{\text{MT}}$) should be used as an over-all measure of performance if different task conditions (other than distance and target width) are compared. This is computed as a mean for each participant and averaged across all participants:

$$\text{TP} = \frac{1}{y}\sum_{y}^{i=1}(\frac{1}{x}\sum_{x}^{j=1}\frac{\text{IDe}_{i,j}}{\text{MT}_{i,j}})$$

   where y is the number of participants and x is the number of movement (distance, width) conditions. The unit of throughput is *bits per second*.

Since the majority of the touch-screen gestures involve some form of dragging, we focus only on dragging-target-acquisition task in this study instead of pointing tasks. To this end, we designed a dragging task where participants are to keep their finger on the screen until they have reached the target.

5.1.2 Occlusion

Occlusion is recognized as a problem in touch screen interactions (*Vogel and Casiez*, 2012), with multiple solutions proposed. For example, finger touch "ghosts" are used for precision pointing on touch screen (*Benko et al.*, 2006). Visuals can be shifted from under the finger/hand to be more visible (*Vogel and Baudisch*, 2007; *Freeman et al.*, 2009). *Cockburn et al.* (2012) also found that occlusion inherent in touch screen pointing can be a limiting factor in performance versus other indirect pointing devices.

However, we are not aware of works quantifying the effect of occlusion in a Fitts's motor performance task on touch screens. Our drag gesture experiment includes potential occlusion as a factor, since for multi-touch gestures it is common to have two fingers or even

two hands over the screen, possibly obscuring the target of the gesture strokes.

5.2 Experimental Design

We recruited 20 adult participants from the University of Michigan. Using a screening survey, we selected participants who were capable of operating a touch screen device, right-handed with right eye-dominance, and who had no impairments that affect the usage of their arms and hands. The effect of eye-dominance on hand motor tasks is not commonly examined, while hand-dominance is found to have a significant effect (*Ehrenstein*, 1997). To avoid both as potential confounding factors we chose to control for both dominances in our participant selection.

Demography of the participants is 50% female and 50% male with an average age of 23.05 years (standard deviation of 2.27 years), and all have some experience using a mobile touch-screen device (smartphone or tablet).

We used full factorial design with the following factors: drag direction (2), drag distances on tablet (4), target size (3), device screen size (2), resulting in 48 conditions in total. With four trials in each condition, a total of 192 trials are presented in randomized order, in 16 batches of 12 each. A full factorial design with randomized order allows straightforward analysis with ANOVA, while the randomization provides counterbalancing to control the effect of presentation order.

For each trial, we asked the participant to place the pointing finger of their dominant hand on a green starting rectangle on the tablet, and then a target is shown on screen. The participant is then asked to drag their finger from the starting position to the target as fast as possible and the trial is complete when the participant lifts their finger. The trial is counted as a success if participant reached the target (lift their finger inside the target), or failure if missed. Audio and visual feedback is provided in all trials to indicate a success or a miss. A crosshair is updated continuously on screen to indicate the center of the touch event as interpreted by the touch screen, and the target square changes color to white when

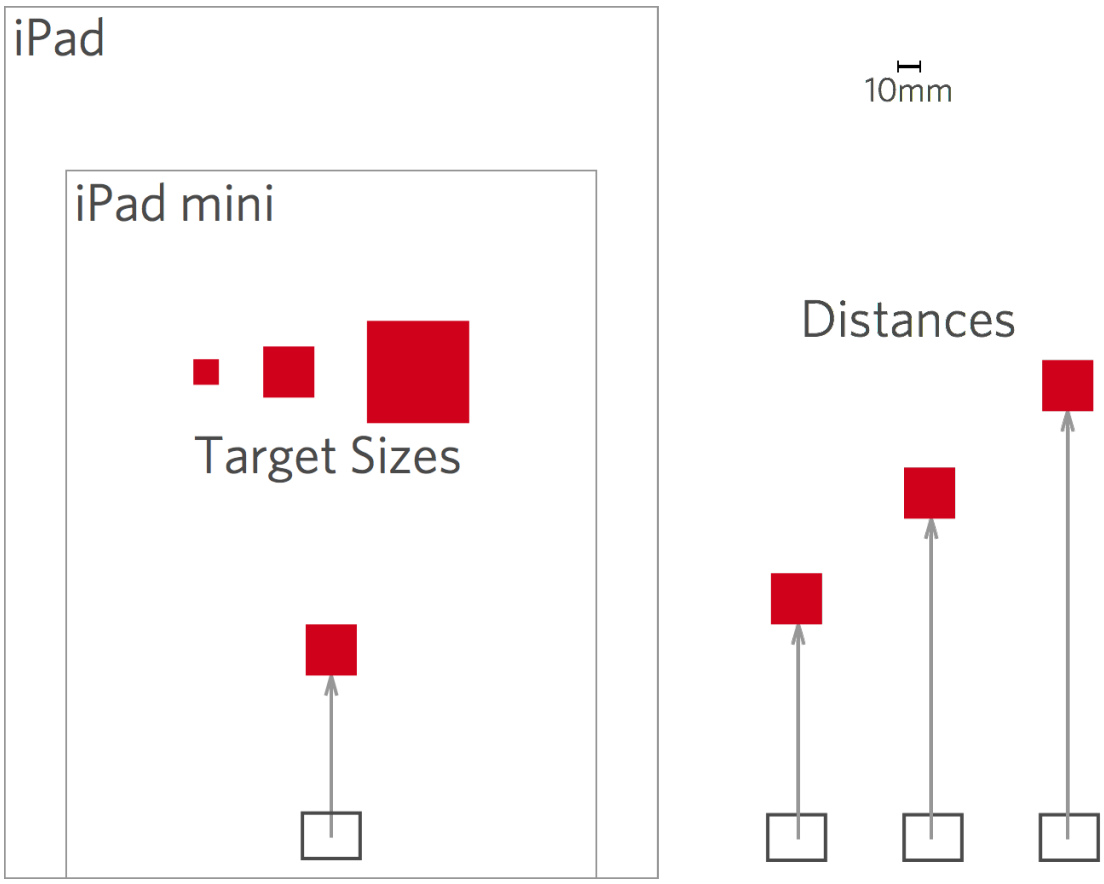

Figure 5.1: Target distances and target sizes in relation to screen sizes, in scale

the participant moves into the target. A short audio beep is played if the participant moved their finger into the target, while a different beep plays when they missed the target. The use of audio and visual feedback for each task is similarly used in other touch-screen Fitts's Law studies (*Bi et al.*, 2013). In our case the visual and audio changes aim to compensate for possible accuracy limits in touch screen gestures, which is not a factor since it is applied to all conditions.

Prior to the actual study, we conducted an informal pilot study to examine factors and to test some assumptions in guiding the experimental design choices. Details of the experiment factors are detailed below:

Drag Direction

We vary the drag direction to introduce potential occlusion as a factor. We positioned

the tablet so that it is vertically aligned with the right forearm of the participant when they lay their forearm flat on a table positioned in front of them. The dragging tasks presented are strictly vertical, dragging either up or down on the tablet (see Figure 5.2). We discovered that diagonal motions with respect to participant's forearms produced potential for limb locking during our pilot study, which can possibly affect the performance. Therefore we choose to make motion vertical and symmetric with respect to forearm joints. With a fixed posture and the alignment of the tablet, when dragging up, the target is above the participant's hand and is always visible with no occlusions. When dragging down, since the participant is moving their hand/arm down vertically , their arm or hand is likely to obscure the drag target.

Target Distance

We vary the distance between the starting position and the center of the target rectangle in range of 37.5mm, 50mm, 75mm and 100mm. The first three distances chosen, mirror previous Fitts's Law studies on touch and pen-based pointing for consistency and comparability (*Graham and MacKenzie*, 1996). The target rectangle is always placed in relation to the initial touch center of the participant's finger, not the center of the starting rectangle, in order to compensate for any variation in each participant's initial touch down. See Figure 5.1 for target distance and size in scale.

Target Size

We use three target sizes: 6mm, 12mm and 24mm. Each target is shown on screen as a red square. The target sizes are chosen in relation to the average human finger width of about 16 – 20mm (*Dandekar et al.*, 2003), with 6mm being close to the typical minimal touch target size on commercial touch-based platforms, 12mm being similar to touch area of average finger, and 24mm giving significantly larger area than the touch area of a finger. Similar to the target distances mentioned above, the values chosen are used in previous Fitts's studies on touch and pen-based pointing

for consistency(*Graham and MacKenzie,* 1996) . The combination of distance and size produces an *ID* range of 1.4 – 4.4.

Device Size

We use two screen sizes, iPad at 9.7" and iPad mini at 7.9". This matches the screen size in the multi-touch programming study in the previous chapter, where we found that smaller screen device perform significantly better (lower task completion time) than the larger screen. Here we want to see if the same effect is in play in a simpler dragging task. Other factors such as target sizes and target distances are scaled according to the different pixel densities of the screens so the physical size and distance remain constant.

For each participant, after introduction and a learning period of about five minutes (96 trials are presented), the batches of trials are presented. In our pilot study, we found that initial variation due to learning or acclimation takes about five minutes to subside, and roughly 100 trials are enough to reduce the effect. Each trial is presented discretely and takes only a couple of seconds, with each batch of trials taking one minute or less to complete. After each batch we ask the participant to take a one-minute rest break to reduce the potential for fatigue. The total time of the session for each participant is about 35 minutes.

5.2.1 Data Collection

We record the time it takes for the participant to move from starting position, into the target, and lift their finger. To compute *Movement Time* (*MT*), we discard any time after the participants first placed their finger on the screen (which triggers the start of the trial, the target is immediately shown), before they moved their finger, so reaction time is not included in *MT*, which is the time participant spend moving their finger while it is in contact with the screen.

We also note when the target is reached or missed for each trial. We instrumented our experiment apparatus to record each drag stroke, since this allows us to record the end-point

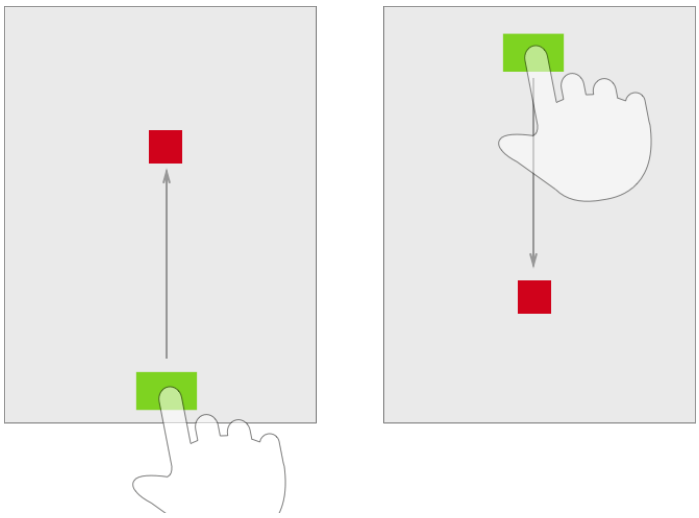

Figure 5.2: Two dragging directions are presented during the dragging task. Participants are expected to experience no occlusion moving up, and more potential for occlusion when moving down.

position of each drag, which are used for *IDe* adjustment later.

5.3 Data Analysis

5.3.1 Data cleaning

Overall we collected a total of 3840 trials from 20 participants. After grouping all data by conditions across participants, we removed a total of 12 outlier samples in each group in preprocessing, (outside of twice standard deviation, similar to suggestion by *Soukoreff and MacKenzie* (2004)), which is less than 0.3% of the total samples.

5.3.2 Movement Time

We use Movement Time (*MT*), the time the participant takes to drag from the starting position into the target square (or stops, if they missed it) as the primary measure of performance. ANOVA on movement time with respect to all factors (i.e., device, direction, target distance and target size) shows that all have significant effect on *MT*. For device size, we found that the iPad mini with smaller screen performed significantly slower than iPad

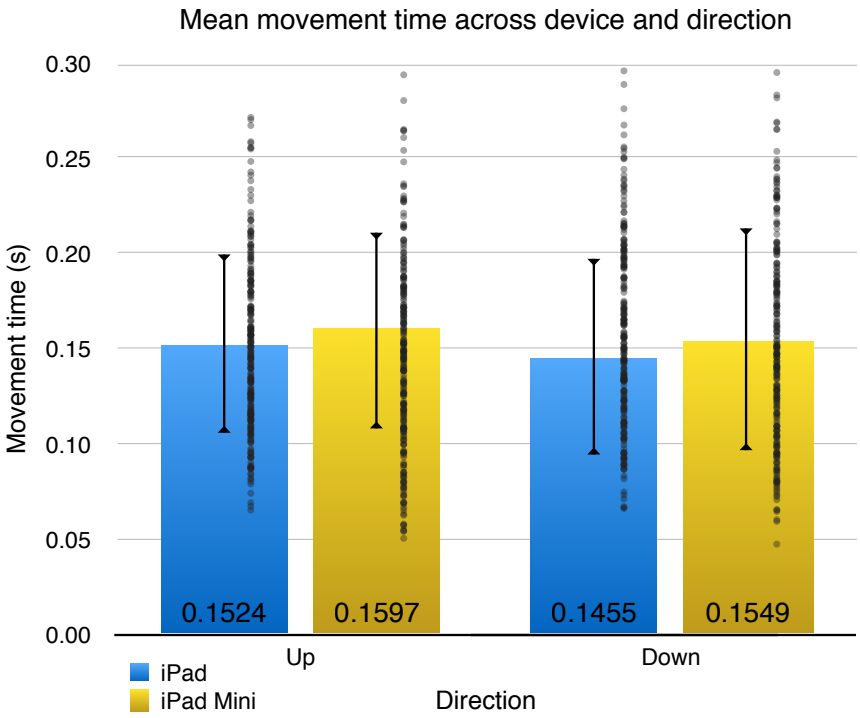

Figure 5.3: Mean movement time are significantly different between device sizes and drag directions. The scatter plot shows the spread of all movement time in each device-direction condition.

with larger screen ($F = 18.93, p < 0.0001$,), which is unexpected given the previous study. We also found direction to be a significant factor ($F = 9.4, p < 0.0022$) (See Figure 5.3, lower time means better performance). For target distance and size this is not surprising ($F = 1299.27, p < 0.0001, F = 641.44, p < 0.0001$ respectively, with $p < 0.05$ as significant), assuming the movement is modeled by Fitts's Law, which predicts a strong correlation between target distance, size and performance. No interaction between device and direction was observed in ANOVA.

5.3.3 Error Rate

We also recorded the number of misses. Because of the scarcity of misses (out of all trials only about 6.1% are missed), we considered error rate per task condition across all participants. In ANOVA with all factors (i.e., device size, direction, target distance, target size) with respect to error rate, we found that drag direction is a significant factor($F = 5.78, p <$

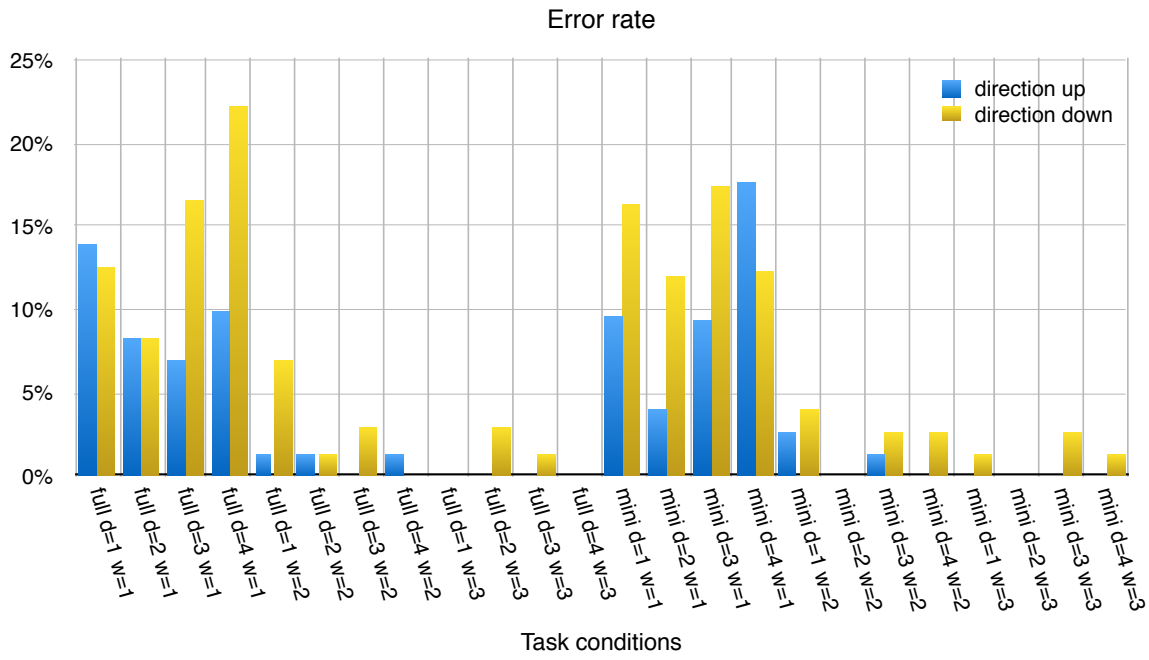

Figure 5.4: Error rate shows significance w.r.t. direction, but not device. As expected target size is significant factor as predicted by Fitts model. Each task condition is labeled by device (full = iPad, mini = iPad mini), distance (d=37.5, 50, 75, 100mm), and target width (w=6, 12, 24mm). Direction up should have no occlusion versus direction down for which occlusion is highly likely.

0.0206), but device size is not (see Figure 5.4). We also found that target width is significant ($F = 83.92, p < 0.0001$) while distance is not ($F = 0.12, p < 0.729$). We did not observe any significant interaction between device and direction in error rate in ANOVA.

5.3.4 Fitts's Law Model

As per recommendation of *Soukoreff and MacKenzie* (2004), using Shannon's formulation of unadjusted *ID*, we first grouped *MT* data into four device size-direction conditions. With *MT* and *ID*, we used least square linear regression to find a, b in MT $= a + b$ IDe. Each condition shows a clear linear fit (see Figure 5.5), with $R^2 = 0.85$ to 0.883.

While unadjusted *ID* assumes participants complete each trial exactly as directed (e.g. stopping in the center of the target), when adjusted for accuracy in real performance, *IDe* should give us a more realistic measure. As noted by *Soukoreff and MacKenzie* (2004), any large discrepancies between the model derived from *ID* and *IDe* may be a sign of a method-

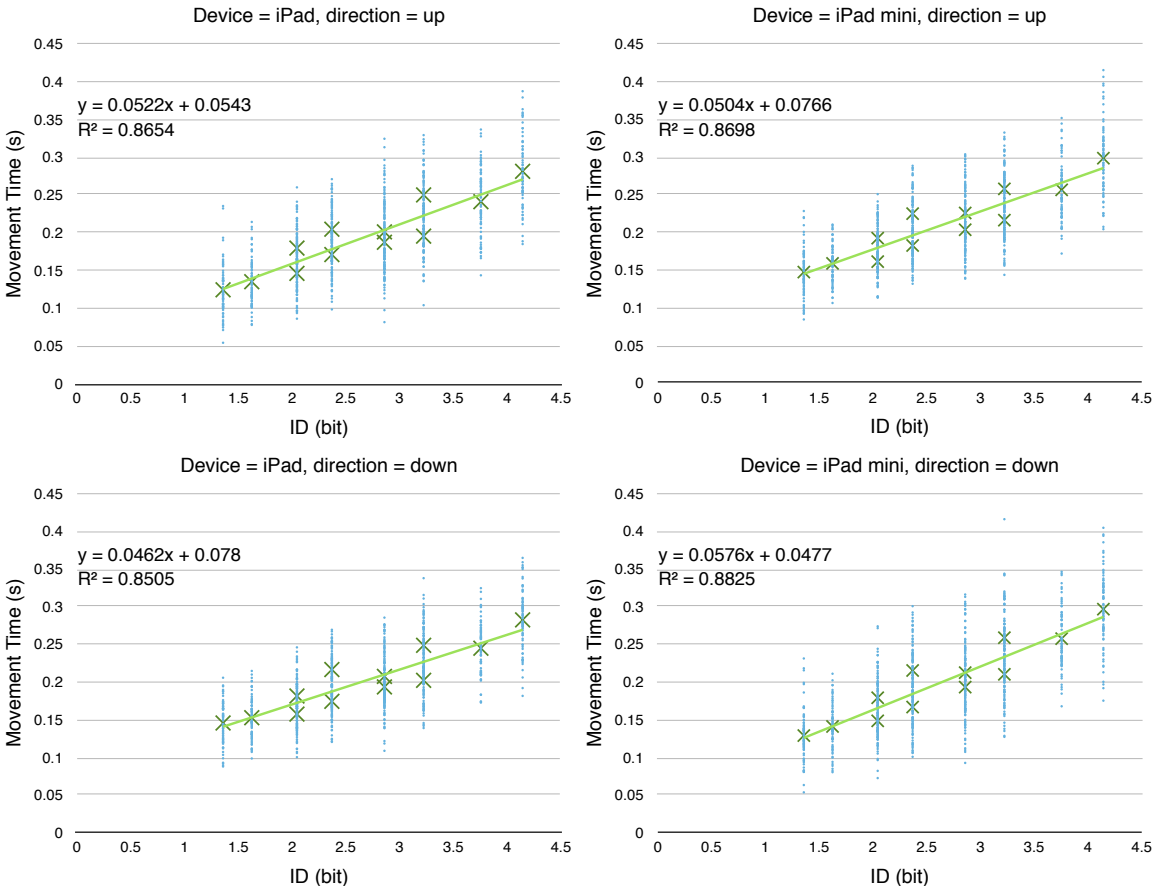

Figure 5.5: Fitts's Law model with unadjusted *ID*, across four device-direction conditions. The spread of *MT* values are also shown in addition to the means for each *ID* values. In all conditions the linear regression fit is shown.

ology problem.

With *IDe* adjusted using collected end-point data, we updated the Fitts's Law model in Figure 5.6. The adjustment does not change the values of *MT*, but shifts the *IDe* values along the x-axis. The linear regression results are similar to unadjusted *ID*, with a clear linear fit $R^2 = 0.824$ to 0.925. In one case R^2 improved from 0.883 to 0.925, while in another R^2 became worse (0.86 to 0.82 after adjustment). Nevertheless, the difference is small, suggesting that Fitts's Law model applies in the dragging task conditions. For both the unadjusted and the adjusted model, the intercept from linear regression is also within the suggested 400ms range at a maximum of 76ms.

To evaluate the performance of the device-direction conditions, we compute throughput

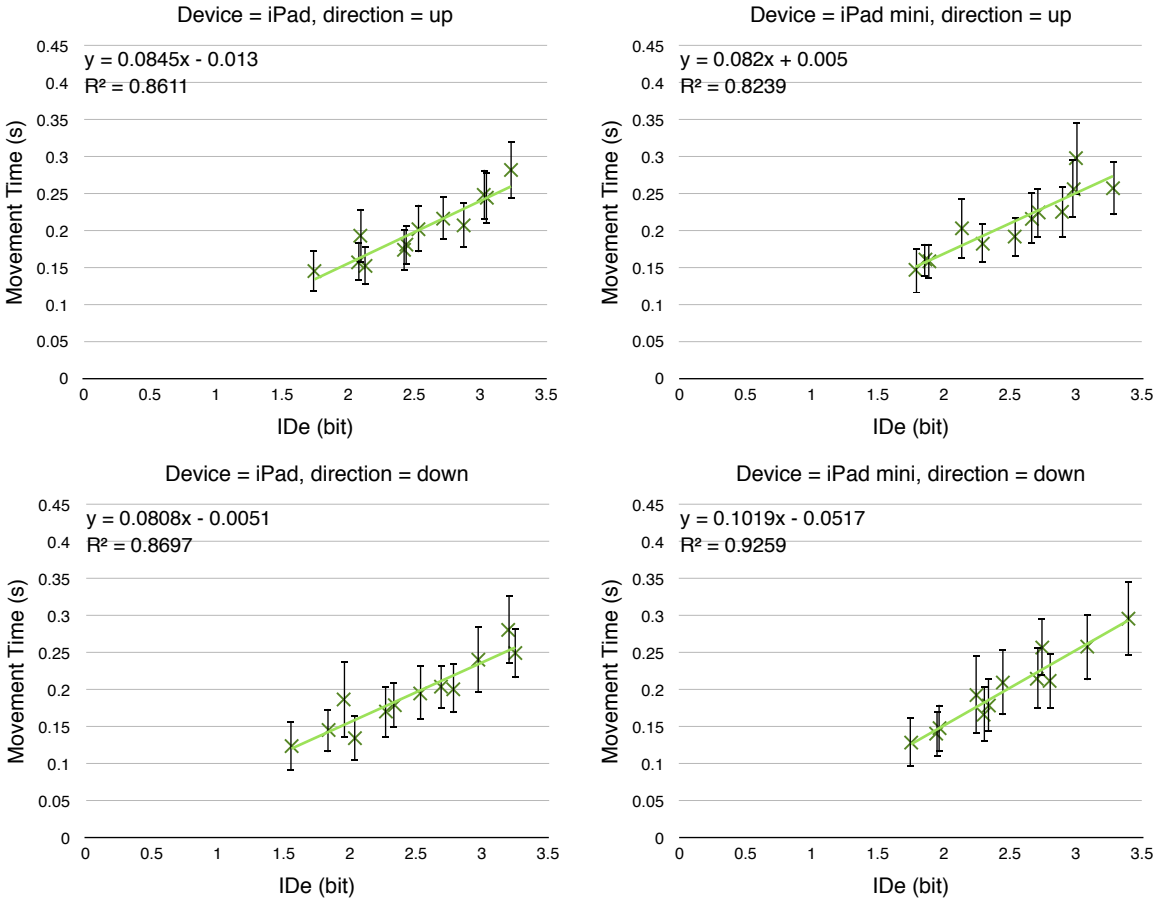

Figure 5.6: Fitts's model with *IDe* adjusted for accuracy across four device-direction conditions. Linear regression fit is also shown for all conditions.

for each participant in each device-size-direction condition (target distance and size are no longer factors). The means of throughput for each condition are similar and range from 12 – 14 bits/s (see Figure 5.7). Running ANOVA on throughput data across all participants (only factors are device size and drag direction) shows that device size is a significant factor ($F = 8.46, p < 0.0037$) and direction is also significant ($F = 11.64, p < 0.0007$). Overall, the iPad mini has lower throughput than the larger iPad, and dragging down leads to higher throughput than dragging up.

5.3.5 Time Effects

To examine the possible effects of time, whether caused by fatigue or learning effects, we looked at throughput in the order of presentation. We use throughput instead of move-

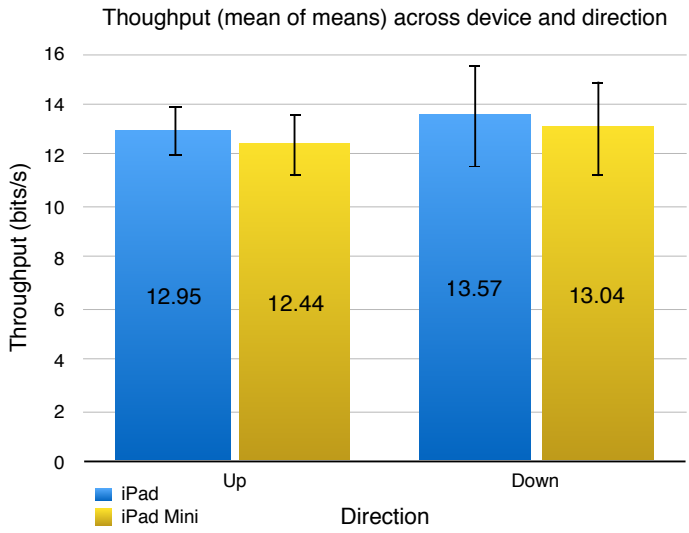

Figure 5.7: Mean of means of throughput per device and direction condition, computed using *IDe*.

ment time due to easier comparability since throughput is a general performance measure independent of target distance and width, which are varied between trials. The aggregated resulting plot shows no significant trend (see Figure 5.8), suggesting no clear effects of learning or fatigue.

5.4 Discussion

In the previous study in Chapter IV, we found that the iPad mini with a smaller screen leads to lower task completion time than the bigger iPad, which corresponds to better performance for smaller screens. Our conjecture was that a smaller screen leads to a lower average drag distance across the tasks performed. If Fitts's Law models drag motion on touch screens, this should lead to lower movement time as well.

Surprisingly, this smaller screen advantage is not reflected in the results. In movement time analysis, the iPad mini has higher movement time than the iPad, suggesting slower performance. Similarly, the iPad mini has a lower throughput, as an overall measure of the performance, while we found error rates are not significantly different across device sizes. This directly contradicts what we expected. Since target distance-size conditions

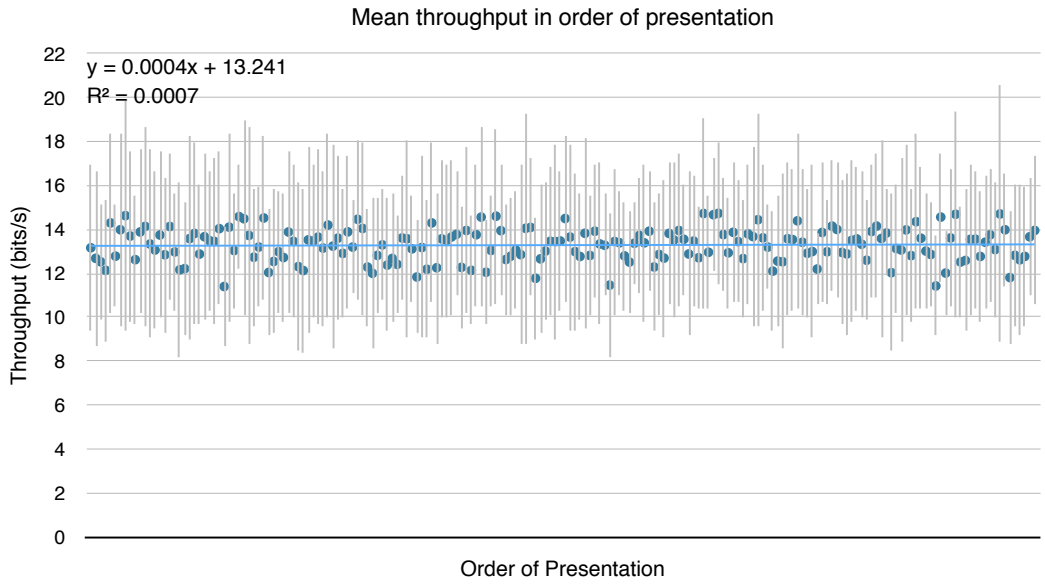

Figure 5.8: Mean throughput shows no significant time effects. Shown are standard deviation of throughput as well as a mostly flat logarithmic regression fit. no learning or fatigue effect significant

are identically scaled between two different screen sizes, if Fitts's Law fully models the task we should expect to see near identical performance across device sizes. We suspect that the counterintuitive result is affected by other potential factors such as perception or confidence of the participant being in play. One potential explanation is that the smaller screen visually outlines a smaller area for the participants to move freely. When asked to drag as fast as possible, their top speed is affected by the perceived freedom to move, as they do not want to move too fast to overshoot the edge of the screen. Larger screens can be perceived to allow more freedom and encourage faster and bigger gestures. In other words, the perception of confinement of the smaller screen potentially makes participants move more carefully, which leads to slower performance overall. Although we did not include a questionnaire in the study, a post-session questionnaire about the participants' perception might help verify or rule out perception as a plausible explanation for this result.

For question (i), we do find that drag motions on touch screens do conform to Fitts's Law model, with both unadjusted *ID* and adjusted *IDe* resulting in good linear regression fit with no large discrepancies.

For question (ii), in the previous study we observed that larger devices lead to longer drag time, however drag distance is not significantly different between devices. The applicability of Fitts's Law model would be potentially consistent with a smaller screen performing better *if* drag distances are also smaller *and* that no other factors are involved, since Fitts's Law predicts that shorter target distance correlates with shorter movement time. However, drag distance is not significantly different in the previous study, even though in this study we see that touch screen drag motions do conform to Fitts's Law. It is less clear if other factors such as perception or experience can also account for the difference in the previous study, which dominate the affect of Fitts's Law drag motions.

Finally, to address question (iii) about occlusion, we find the potential presence of occlusion to be a significant factor across multiple measures in unexpected ways. We expected downward drag to have potentially more occlusion, which might affect performance negatively. Instead, movement time is significantly lower when dragging down, suggesting higher performance. Similarly throughput measure shows downward drag lead to higher performance. A potential explanation is that the reduced visibility of the target due to occlusion leads to a less careful approach to the target, which may lead to faster movement. This explanation is consistent with what we see in error rates. Although errors are sparse, when participants drag down, they miss significantly more than when they drag up. The higher error rate (especially when the target is small) supports the explanation that potential occlusion in downward drag leads to less careful drag approaches. Overall, the result is that participants tend to move faster during downward drag but are less accurate.

5.5 Conclusion

As an extension of the work in Chapter IV, we conducted a Fitts's Law drag motion study to quantify the performance difference in touch screen drag between device sizes and the potential effect of occlusion. We find Fitts's Law to suitably model drag motion on touch screen overall, but the difference across device sizes is unexpected. Smaller screens lead to

slower dragging motion and lower throughput, while in our previous study smaller screens lead to higher performance on complex tasks. We suspect factors such as perception and confidence of the participant with respect to screen size might be in play. We also quantified the difference due to potential occlusion in dragging motion, and found that potential occlusion is correlated to a higher performance (i.e., lower time, higher throughput), but also higher error rate.

CHAPTER VI

Tapperware: Implementation

The mobile visual programming system called *Tapperware* is used in **Chapter IV**. Tapperware is built on top of the urMus framework (*Essl*, 2010b) using Lua scripting language. In this chapter, we present the implementation detail of Tapperware, and provide some examples of how it can be used to create gesture interfaces with visualizations.

6.1 Overview

The architecture of Tapperware (see Figure 6.1) consists of a set of programming primitives accessible through an Application Programming Interface (API), and the graphical interface components that enable the interaction and feedback. The programming primitives include classes that represent the basic entities in the visual language (see **Chapter IV**): *Region*, *Link* and *Group*. These classes mirror their syntactic counterparts in the visual language. The graphical representations of these primitives and the user interfaces are managed separately by a set of singleton classes, which also provide standard UI widgets such as menus, icons, notification views, and visualization for gestures. Finally, a set of utility classes provide a logging feature which can be used for user studies, and an interface to urMus's sound synthesis API.

We use Lua's general data structure `table` to achieve object-oriented programming. This enables common object-oriented behaviours such as inheritance, which we will not

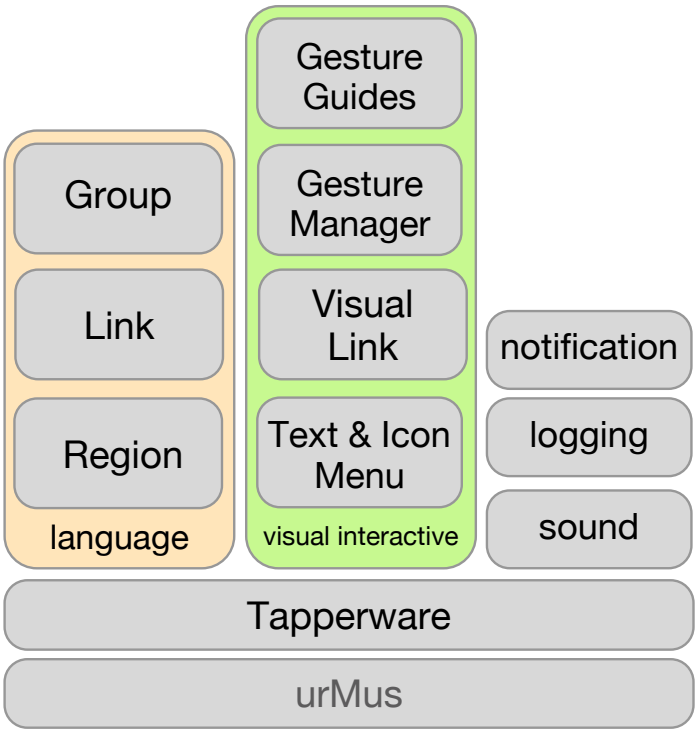

Figure 6.1: Overview of the software components of the programming environment

describe in detail here[1]. Syntactically, if `a` is an object, then `a:method()` is a method call, and a property of object `a` is accessed as `a.property`.

6.2 Programming Primitives

6.2.1 Region

Tapperware `region` is an extension or subclass of the built-in visual region class in urMus, as it retains appearance properties and direct-manipulation interactions. The urMus region API[2] provides simple access to the creation of an onscreen rectangle, as well as the modification of its texture, and event-driven interaction using callback functions. For example, the following code creates an *urMus region* and then configures its size and position on screen:

```
r = Region()    -- creates a region object
r:SetWidth(200) -- sets dimensions and positions
```

[1] See Lua documentation at `http://www.lua.org/pil/16.html` for detail
[2] `http://urmus.eecs.umich.edu/urAPI/Region.html`

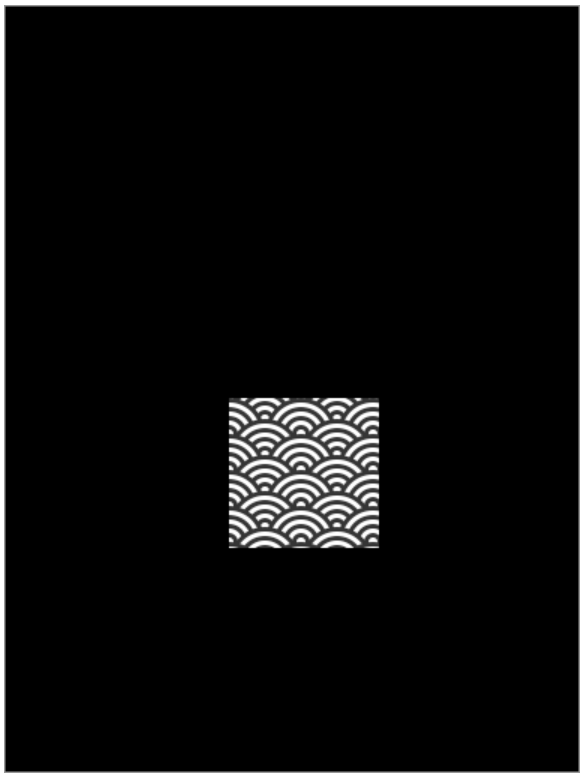

Figure 6.2: An urMus region with size, position and texture configured

```
r:SetHeight(200)
r:SetAnchor('CENTER',400,400)
r:Show()        -- make it visible on screen
```

The appearance of the urMus region can also be modified, either by setting the color or, as shown in the following example, setting an image texture:

```
r.t = r:Texture("textureImage.png")
r.t:SetBlendMode("BLEND")
```

In this example, `r:Texture()` method call initiates a texture object and is then stored in `r.t` property, and Figure 6.2 shows the final result. Texture API [3] in urMus allows a variety of visual manipulations such as setting the blend mode.

[3] `http://urmus.eecs.umich.edu/urAPI/Texture.html`

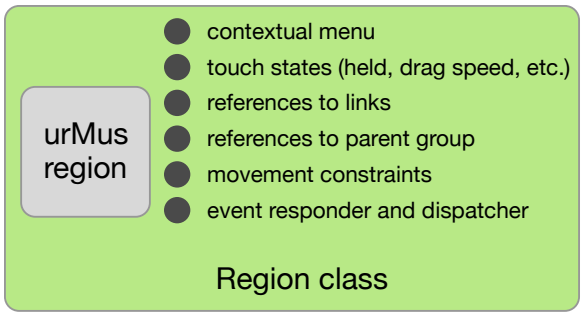

Figure 6.3: Tapperware `region` class extends *urMus region* class with multiple properties and mechanisms as part of the gesture state machine.

Event-driven logic can be set up in urMus regions using `Handle()` methods:

```
-- define the call back function

function onTouchUpCallback(self, x, y)

        Log:print("touched up on region")

end

-- now assign the call back function

r:Handle("OnTouchUp", onTouchUpCallback)
```

Here the function `onTouchUpCallback` is passed as the callback function for a low level touch event, and will be called whenever a user triggers a touch up event in the urMus region `r`.

In Tapperware, the extended `region` class retains all the features of the urMus region class mentioned above. In addition, it also stores interaction states as part of the gesture state machine, and mechanisms for dispatching and responding to events. Events and Links (described in the next section) are the main mechanisms through which `regions` interacts with each other. It also has data structure for managing links and associated contextual interfaces for text and icon-based menus (see Figure 6.3). A highlight of the new public methods include:

`TWRegion:new()`

A new constructor for the extended `TWRegion` class, new `TWRegion` objects are either allocated or used from a reusable pool for more efficient memory usage.

`TWRegion:RemoveRegion()`

A recycling destructor which returns the TWRegion object into a reusable pool. This will also remove all the links to and from this region.

`TWRegion:SetPosition(int x, int y)`

Sets the onscreen position of the object, while obeying any movement constraints. For example, if the region object is pinned or is within a group, the new position is set according to the constraints.

`TWRegion:Copy(x, y)`

Creates a copy of the region object and positions it onscreen. This also duplicates any links associated with the object, mirroring the behaviour described in **Chapter IV**.

`TWRegion:OpenRegionMenu(), :CloseMenu()`

Opens or closes the associated contextual menu. This could be a text-based menu or an icon-based menu depending on the menu class used.

Tapperware `region` has event handlers for all of the low level touch events in urMus, and each event is passed to the `gesture manager` class as part of the gesture recognizer, which we will discuss in Section 6.3.1.

6.2.2 Link

The function of `Link` class is to connect two `region` objects and to direct the flow of events from object to object (with a specified sender and receiver, see Figure 6.4). Each `link` object can also have user interfaces associated with it (for example, buttons for removing the link). Methods of `link` class include:

`Link:new(TWRegion sender, TWRegion receiver, event, eventHandler)`

Constructor for link object, which also sets the `sender` and `receiver` regions. `Event`

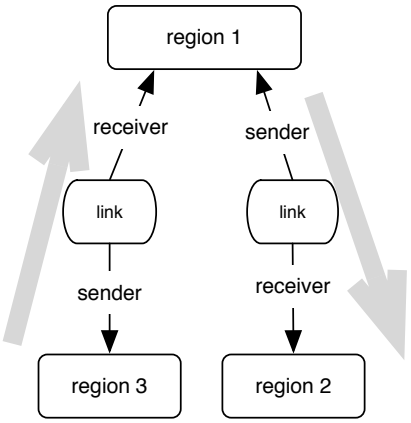

Figure 6.4: The `link` class is a directional conduit between region objects. The large grey arrows indicates the flow of events.

is a string specifying the type of events to be passed through the link, and `eventHandler` is a reference to the callback function, which usually belongs to the region object.

`Link:SendMessageToReceivers(message, TWRegion origin)`

This private method is called only by the `sender` region object to pass the event to the `receiver`. `Message` can be of any types as long as the handler function knows how to process it. `Origin` references to the `sender` object, which is used to detect cycles when a message is propagated through multiple regions connected by links. A simple tree traversal is used to send each message, where regions are nodes and links are edges.

In the following example, two region objects are linked so that they will move together (touch event `OnDragging` is handled with the region method `TWRegion.move`):

```
r1 = TWRegion:new()
r1:SetPosition(10,10)
r2 = TWRegion:new()
r2:SetPosition(20,30)
```

```
link:new(r1,r2,'OnDragging',TWRegion.Move) -- link from r1->r2
link:new(r2,r1,'OnDragging',TWRegion.Move) -- link from r2->r1
```

6.2.3 Group

`Group` is a subclass of the `region` class, which acts simply as a container for other `region` objects. It has common methods for managing its child regions:

`Group:New(), Group:Destroy()`

Constructor and destructor for group object (The destructor also removes any regions contained in the group.)

`Group:CreateGroupFromRegions(listOfRegions)`

Special constructor to create a new group to contain a list of regions

`Group:SetRegions(listOfRegions), :AddRegion(region), :RemoveRegion(region)`

Methods for setting, adding regions to a group, as well as removing regions from a group

Like `region` objects, `group` objects can also be connected by `links`, or duplicated. In the following example code, two regions are created, and are then nested together with a new group.

```
local r1 = TWRegion:new()
local r2 = TWRegion:new()

rgroup = CreateGroupFromRegions({r1,r2})
rgroup.r:LoadTexture('barback.png') -- change group's texture
```

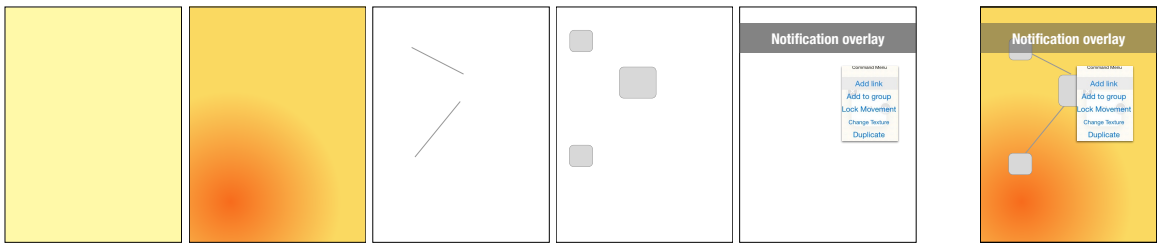

Figure 6.5: Illustration of the different layers which are composited together to form the graphical interface of Tapperware

6.3 Graphical User Interface

The visual representation of `region` objects is handled by urMus region API, while the visual representation of `links` and other user interface elements are built from urMus regions. The interface of the environment is composited from several layers of urMus regions, each responsible for drawing the background canvas, links, each `region`, menus and icons, gesture visualizations, and any notification overlays (see Figure 6.5). Multiple utility classes are responsible for handling the drawing of these layers:

`LinkLayer`

This class draws each of the line segments on screen representing the individual link objects.

`SimpleMenu, IconMenu`

These classes create and manage a text-based or icon-based menu on screen (shown in Figure 4.5 and Figure 4.7). In addition to drawing all the icon textures, icon menu class also handles the drawing of selection lassos as seen in Figure 4.8.

`ImgPicker`

This class creates an image-based texture-picker menu on screen (see Figure 4.4).

`NotifyView`

This class creates a simple timed text-based notification, and can be used for feedback when actions are performed.

`GestureGuideView`

This class contains all drawing methods for different types of visual gesture guides, which are called by the gesture manager/recognizer for feedback.

The text and icon menu classes are implemented using textured urMus regions and text labels. They are designed to be reusable and are instantiated and configured each time they are needed with callback functions for each command in the menus. The drawing and updating of links and gesture guides are triggered by user-initiated events such as the moving or tapping of a region. These low level touch events are handled through a `gesture manager` utility class.

6.3.1 Gesture Recognition and Visualization

Gesture interactions are enabled by two components, the `Gesture Manager` class which handles the actual recognition of interactions using state machines, and the `GestureGuide–View` class which draws the visualization for gestures on screen. The multi-touch gesture state machine recognizes each gesture such as a pinch by listening for low-level touch events from all regions in the environment, and then triggers the assigned actions (for example, creating a link) when action states are reached. To accomplish this, `Gesture Manager` listens for low level touch events using the following callback functions which are called by `TWRegion` when low level touch events are received:

`GestureManager:BeginGestureOnRegion(region)`

Called when a touch-down event is detected on any region. The gesture manager keeps track of the number of concurrent touch-down events to differentiate between multi-touch and single touch gestures.

`GestureManager:EndGestureOnRegion(region)`

Called when a touch-up is detected on any region

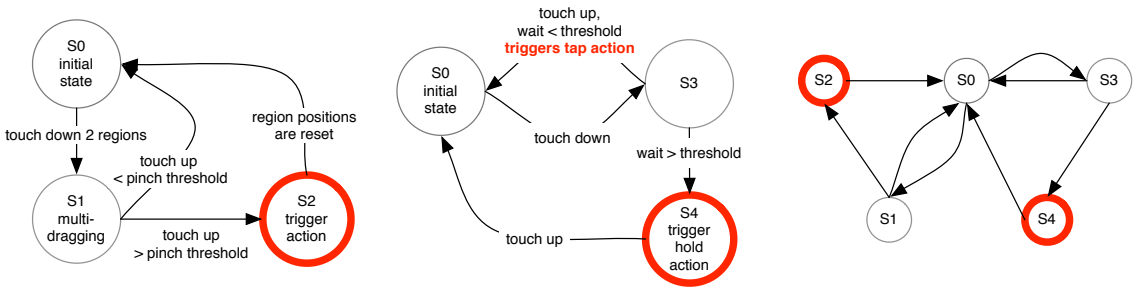

Figure 6.6: Simplified examples of state machines managing the pinching gesture, tap and hold gesture. On the far right is a combined state machine for both the pinching and hold gestures. Action states are shown with bold, red circle and text.

`GestureManager:Dragged(region, dx, dy, x, y)`

Called when an `OnDragging` event is detected, when the region is being moved by the user. The `dx, dy` reports the velocity of the drag motion so that the manager can filter on intentional drag versus unintentional shift in touch when the user intended to tap and hold instead.

`GestureManager:Tapped(), :Leave()`

These callback methods are triggered when a `tapped` or `leave` touch event is fired by user input.

Since low level events such as *touch-down*, *touch-up*, *tap*, and *leave* are recorded by the `gesture manager`, higher level multi-touch or single touch gestures can be specified through a gesture state machine. For example, Figure 6.6 shows two simplified state machines for a two-finger pinch gesture, as well as tap and hold gestures. When each gesture state is entered, drawing functions in `GestureGuideView` mentioned above are called accordingly to provide and update the visualizations. Some visualization is updated after each `GestureManager:Dragged()` call to provide continuous visual feedback.

The separation of gesture recognition and visualization in the `Gesture Manager` class allows a variety of gestures to be swapped in by specifying different state machines within the manager class, and different visualizations to be exchanged by only changing the implementation of `GestureGuideView`. This way the underlying functional component such as

region and link objects do not need to be modified for new gestures or new visualizations. However, one particular technical challenge is the limitation in expressivity and readability of specifying complex state machines in code. While the example presented in Figure 6.6 is simplified, more complex gestures might call for different approaches to programming state machines such as using specialized or declarative specifications.

6.3.2 Utilities

Other utilities include logging, which records each touch event and user-triggered action in a detailed log on device. This utility is used extensively in **Chapter IV** for data collection during the user study.

In addition to the visual interface, a sound module serves as a simple wrapper to urMus sound synthesis API. This allows user inputs from the programming environment to be fed into sound synthesis parameters, and enabling the building of musical controllers. In the example in Figure 4.14, each of the sliders controls a different parameter of a simple sin oscillator.

6.4 Summary

In this chapter we presented an overview of *Tapperware*, a multi-touch visual programming environment which can also support the creation of new multi-touch gesture interfaces. We described the architecture of Tapperware, and a summary of the Application Programming Interfaces for the programming primitives: Region, Link and Group. Finally we outlined the components for gesture recognition and gesture visualization and how they can be independently configured to create new gesture-based interfaces.

CHAPTER VII

Conclusion

The work we presented in this thesis investigates the interrelation between visual representation and feedback, and gesture-based input in the domain of creative computing. While previous works have looked at the mechanics for the recognition of gesture input and the design of visual feedback, we focused on the joint consideration of gesture interfaces and visualizations across a range of interaction spaces.

Our research question is: *Which aspect(s), if any, of visual feedback and sonic mapping affects the efficacy of gesture-based interaction in creative computing?* We examined a range of interaction spaces and usage scenarios, from continuous sound parameter control in open-air gestures to visual programming on multi-touch mobile devices. The main finding of this thesis is that performance in gesture interactions is dependent on multiple and interacting factors. In other words, *not all gestures are created equal*.

We presented two novel gesture interaction spaces, an open-air gesture-augmented piano keyboard, and a multi-touch visual programming environment. We conducted user studies and found that user performance is dependent on multiple, interacting factors such as the mapping of gesture to sound and device size. A number of findings are unintuitive. For example, the contradictory effect of device size in two different task scenarios. This suggests that the performance of similar gestures, such as dragging, can vary highly based on the scenario and may benefit from different task and conditions like device size.

7.1 Contribution

Specifically, we set out to accomplish the following goals:

1. Examine the role of visual representation and mapping in real-time continuous gesture-to-input-parameter interaction, and discrete gesture-driven commands.

2. Quantify user performance in gesture input across a range of interaction spaces to examine the effect of multiple factors on gesture interactions.

3. Develop tools and platforms for exploring visual representations of gestures.

Goal 1

In **Chapter III** and **Chapter IV**, we considered visualization and gesture interface in a range of interaction spaces. We explored the design space of visual representation and feedback in both continuous gesture parameter controls and discrete gesture-driven visual programming. In the space of open-air gesture controls, we defined a framework for design visuals and gestures based on physical placement and temporal-causal relationships between the input and the feedback. We created a series of visualizations and representations to explain and guide user interactions by drawing on real-world metaphors such as a school of fish or a harp. We also considered the design of visual representations in multi-touch visual programming. We created multiple visual representations, from a menu-driven interface to a direct-manipulation-inspired gesture interface. This work is described in **Section 3.4** and **Section 4.3.2**.

Goal 2

We presented three user studies which quantified the performances of gesture interfaces with respect to varying mapping strategies, visual representations and device sizes.

In **Chapter III** we examined the performance of real-time continuous parameter gesture controls in a gesture-augmented keyboard instrument, using quantitative measures such as task completion time and smoothness of input in a real-time musical performance

scenario. We found that the mapping of gestures to multiple continuous parameters affects performance significantly. The best gesture-parameter mapping can out-perform traditional physical wheels for real-time multi-parameter control. For example, hand-width gesture is more suitably matched with tremolo sound control, but performs poorly when mapped to a low-pass filter effect. This advantage is also correlated with the subjective measure of higher expressivity and more enjoyment in using the gesture interface controls over traditional physical controls. In summary, we found that not all gesture-to-sound mappings are equally suited for performance.

In **Chapter IV** we looked at the performance of three visual representations and two device sizes in the context of mobile multi-touch visual programming. We conducted a user study and measured performance with metrics such as task completion time, waiting time and drag time. We found that more traditional menu-driven and icon-driven visual representations achieved better performance than a gesture-driven representation. While users reported that menu and icon-driven representations are easier to use, gesture and icon representations are more enjoyable to use. We also unexpectedly found that smaller device size, which corresponds to smaller screen size, leads to better performance overall, which is possibly correlated with overall distance of drag gestures. This performance difference in device size is further investigated in a drag task study using Fitts's Law in **Chapter V**. We found a counterintuitive result in touch screen drag tasks which challenges our earlier findings. We found that smaller device size actually leads to worse performance in speed, possibly due to a difference in user perception of confinement on the device. We also unexpectedly found that the potential of occlusion in the task condition does not lead to slower performance. Instead users performed the drag task faster, but with a lower accuracy. These results suggest that not all touch screen drag gestures are the same, and performance is dependent on factors such as device size and the scenario.

In both gesture spaces, we found that the performance of gesture interactions is dependent on multiple interacting factors, and the effect of these factors is also dependent on the

scenario and the task.

Goal 3

We present *Tapperware*, a multi-touch visual programming platform in **Chapter VI**, which can serve as a starting point for exploring multi-touch gesture interfaces and visualizations. This framework is used as the basis for the user study in **Chapter IV**. It is designed to allow new multi-touch gestures to be defined using state machines as well as different gesture visual feedback to be implemented independently.

Further contributions made in this thesis are as follows:

Gesture Augmented Musical Keyboard Instrument

In **Chapter III** we presented a novel gesture-augmented musical keyboard instrument where gestures are used to control continuous sound parameters. The design of the instrument combines open-air gesture sensing with a traditional piano keyboard, as well as top-down projection for visual feedback over the entire gesture space. The gesture sensor can track hand positions in 3D space and can provide corresponding audio output and visual feedback overlaid on the gesture space. This system serves as the platform for our user studies in comparing real-time gesture mapping and physical wheel controls. It also enables a range of alternative performance interfaces. For example, the player can move either hand into the gesture space immediately above the keyboard for parameter controls and using the other hand to continue playing simultaneously. The gesture space and projected visualization open up many possibilities of musical instrument designs. Some examples can be found in **Section 3.4**.

Mobile Visual Programming with Multiple Visual Representations

In **Chapter IV** we presented a visual touch-driven environment for interactively constructing dynamic interfaces. We created three distinctive visual representations for a common underlying flow-based visual programming vocabulary: a traditional menu interface, an icon-driven interface, and a gesture-driven interface. While menu interface follows more established text-based menu paradigm, icon and gesture repre-

sentations make use of novel visual metaphors and guides for continuous feedback. As detailed in **Chapter VI**, this environment is also designed to be extensible for prototyping other gesture interfaces.

7.2 Future Research

In the short period of time since multi-touch and open-air gesture interfaces have become common on commercial platforms such as smartphones and gaming consoles, there already have been many approaches in gesture interfaces. We believe the design of gesture interaction in the creative domain is a growing area ripe for exploration and experimentation. Through our exploration of visualization and gesture interaction design, we have started to develop a framework to understand the interaction between visuals and gesture input, as well as guiding the design of new gesture interfaces. From our user studies, we have seen that certain gesture mappings and visual representations have performance advantages over others. The precise mechanism of these advantages is not yet clear understood. For example, the suitability of certain mapping of gesture to sound parameter suggest a possible cognitive fit between the two factors, more research is needed to understand this relationship.

The various factors involved in multi-touch gestural interaction also call for further investigation. Although factors such as occlusion and device size are recognized, and other works have proposed solutions to these perceived problems, more studies are needed to examine *how* these factors affect the interaction. In our examination of device size and occlusion, the counter-intuitiveness of the results suggests that the mechanisms of their effects are not well understood. For example, it is not conclusive or clear why in one scenario users performed better on smaller devices, while in another they performed worse. A deeper understanding of these factors can help guide the design of gesture interactions to avoid any pitfalls.

With the growing adoption of personal mobile devices, demand is also growing for cre-

ative and personal expressive use of these devices. We are in an exciting time when the landscape of gesture interactions is shifting with new device form-factors and commoditized gesture sensing technologies. These developments point to a hopeful outlook for gesture interactions where expressivity and enjoyment can rival traditional physical musical instruments, while their specificity and power can match the complexity afforded by the keyboard-and-mouse paradigm.

BIBLIOGRAPHY

BIBLIOGRAPHY

(2002), Ergonomic requirements for office work with visual display terminals (vdts)—part 9—requirements for non-keyboard input devices, (ISO 9241-9:2000(E)).

(2011), Appendix D Assessment of Comfort, *Ergonomics of human-system interaction - Part 420: Selection of physical input devices.*, (ISO 9241-420:2011), 38–30.

Apple Inc. (2013), iOS Human Interface Guidelines, Apple Inc.

Bark, K., E. Hyman, F. Tan, E. Cha, S. A. Jax, L. J. Buxbaum, and K. J. Kuchenbecker (2013), Effects of Vibrotactile Feedback on Human Learning of Arm Motions, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*.

Bau, O., and W. E. Mackay (2008), Octopocus: A dynamic guide for learning gesture-based command sets, in *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pp. 37–46, ACM.

Benko, H., A. D. Wilson, and P. Baudisch (2006), Precise selection techniques for multi-touch screens, in *CHI '06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, p. 1263, ACM, New York, New York, USA.

Berg, T., D. Chattopadhyay, and M. Schedel (2012), Interactive Music: Human Motion Initiated Music Generation Using Skeletal Tracking By Kinect, in *Proceedings of the Conference of The Society for Electro-Acoustic Music in the United States (SEAMUS)*.

Berland, M., T. Martin, T. Benton, and C. Petrick (2011), Programming on the move: Design lessons from IPRO, in *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pp. 2149–2154, ACM.

Berthaut, F., M. Marshall, S. Subramanian, and M. Hachet (2013), Rouages: Revealing the Mechanisms of Digital Musical Instruments to the Audience, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Bi, X., Y. Li, and S. Zhai (2013), FFitts law: modeling finger touch with fitts' law, in *CHI '13: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM.

Biegel, B., J. Hoffmann, A. Lipinski, and S. Diehl (2014), U can touch this: touchifying an IDE, in *CHASE 2014: Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, ACM.

Blackwell, A., and N. Collins (2005), The Programming Language as a Musical Instrument, *Proceedings of PPIG05 (Psychology of Programming Interest Group)*, pp. 120–130.

Bonnel, A.-M., and E. R. Haftser (1998), Divided attention between simultaneous auditory and visual signals, *Attention, Perception, & Psychophysics*, *60*(2), 179–190.

Bragdon, A., A. Uguray, and D. Wigdor (2010), Gesture play: motivating online gesture learning with fun, positive reinforcement and physical metaphors, in *ACM international conference on interactive tabletops and surfaces*, pp. 39–48, ACM, Saarbrü cken,Germany.

Buxton, B. (1991), The "Natural" Language of Interaction: A Perspective on Non-Verbal Dialogues, *The Art of Human-Computer Interface Design*, pp. 405–416.

Buxton, B. (2007), Multi-Touch Systems That I Have Known and Loved, *Microsoft Research*.

Buxton, B. (2010), 31.1 : Invited Paper: A Touching Story: A Personal Perspective on the History of Touch Interfaces Past and Future, *SID Symposium Digest of Technical Papers*, *41*(1), 444.

Buza, K. (2012), ScriptKit.

Chae, M., and J. Kim (2004), Do size and structure matter to mobile users? an empirical study of the effects of screen size, information structure, and task complexity on user activities with standard web phones, *Behaviour & Information Technology*, *23*(3), 165–181.

Chang, F., C.-J. Chen, and C.-J. Lu (2004), A linear-time component-labeling algorithm using contour tracing technique, *Computer Vision and Image Understanding*, *93*(2), 206–220.

Chaparro, B., B. Nguyen, M. Phan, S. A, and J. Teves (2010), Keyboard Performance: iPad versus Netbook, in *Usability News*, vol. 12.

Charbonneau, E., A. Miller, and J. J. LaViola Jr (2011), *Teach me to dance: exploring player experience and performance in full body dance games*, 43 pp., ACM.

Cockburn, A., D. Ahlström, and C. Gutwin (2012), Understanding performance in touch selections: Tap, drag and radial pointing drag with finger, stylus and mouse, *International Journal of Human-Computer Studies*, *70*(3), 218–233.

Culjak, I., D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek (2012), A brief introduction to OpenCV, in *MIPRO, 2012 Proceedings of the 35th International Convention on Information and Communication Technology, Electronics and Microelectronics*, pp. 1725–1730, IEEE, Opatija, Croatia.

Dandekar, K., B. I. Raju, Srinivasan, and M. A (2003), 3-D Finite-Element Models of Human and Monkey Fingertips to Investigate the Mechanics of Tactile Sense, *Transactions of the ASME*, *125*, 682–691.

Davidson, P. L., and J. Y. Han (2006), Synthesis and control on large scale multi-touch sensing displays, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pp. 216–219, Paris, France.

diSessa, A. A., and H. Abelson (1986), Boxer: a reconstructible computational medium, *Communications of the ACM, 29*(9).

Eaton, J., and R. Moog (2005), Multiple-touch-sensitive keyboard, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Vancouver, BC, Canada.

Ehrenstein, W. H. (1997), Perception-action compatibility and eye-hand dominance in using visually-displayed information, in *Proceedings of the Europe Chapter of the Human Factors and Ergonomics Society Annual Conference*, Bochum.

Essl, G. (2010a), Mobile phones as programming platforms, in *Proceedings of the First International Workshop on Programming Methods for Mobile and Pervasive Systems*, Pervasive, Helsinki.

Essl, G. (2010b), UrMus – An Environment for Mobile Instrument Design and Performance, in *Proceedings of the International Computer Music Conference (ICMC)*, Stony Brooks/New York.

Essl, G., and A. Müller (2010), Designing Mobile Musical Instruments and Environments with urMus, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Sydney, Australia.

Essl, G., and S. O'Modhrain (2006), An enactive approach to the design of new tangible musical instruments, *Organised Sound, 11*(03), 285–296.

Fitts, P. M. (1954), The information capacity of the human motor system in controlling the amplitude of movement., *Journal of Experimental Psychology, 47*(6), 381–391.

Forlines, C., D. Wigdor, C. Shen, and R. Balakrishnan (2007), Direct-Touch vs. Mouse Input for Tabletop Displays, in *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM.

Freed, A., and R. Avizienis (2000), A new music keyboard featuring continuous key-position sensing and high-speed communication options, in *Proceedings of the International Computer Music Conference (ICMC)*, Berlin, Germany.

Freeman, D., H. Benko, M. R. Morris, and D. Wigdor (2009), ShadowGuides: visualizations for in-situ learning of multi-touch and whole-hand gestures, in *ITS '09: Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ACM.

Ghomi, E., S. Huot, O. Bau, M. Beaudouin-Lafon, and W. E. Mackay (2013), Arpège: Learning multitouch chord gestures vocabularies, in *Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces*, pp. 209–218, ACM.

Goel, M., L. Findlater, and J. Wobbrock (2012), WalkType: using accelerometer data to accomodate situational impairments in mobile touch screen text entry, in *CHI '12: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM.

Google Inc. (2013), Metrics and Grids | Android Developers.

Graham, E. D., and C. L. MacKenzie (1996), *Physical versus virtual pointing*, 292–299 pp.

Green, T. R. G., and M. Petre (1996), Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework, *Journal of Visual Languages & Computing*, *7*(2), 131–174.

Grudin, J. T. (1983), Error Patterns in Novice and Skilled Transcription Typing, in *Cognitive aspects of skilled typewriting*, pp. 121–143, Springer New York, New York, NY.

Haken, L., and E. Tellman (1998), An Indiscrete Music Keyboard, *Computer Music Journal*, *22*(1), 30–48.

Hall, A. D., J. B. Cunningham, R. P. Roache, and J. W. Cox (1988), Factors affecting performance using touch-entry systems: Tactual recognition fields and system accuracy., *Journal of Applied Psychology*, *73*(4), 711–720.

Hamilton, R., J. Smith, and G. Wang (2011), Social Composition: Musical Data Systems for Expressive Mobile Music, *Leonardo Music Journal*, *21*(21), 57–64.

Han, J. Y. (2005), Low-cost multi-touch sensing through frustrated total internal reflection, in *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pp. 115–118, ACM.

Hils, D. D. (1992), Visual languages and computing survey: Data flow visual programming languages, *Journal of Visual Languages & Computing*, *3*(1), 69–101.

Hofmeester, K., and J. Wolfe (2012), Self-revealing gestures: teaching new touch interactions in windows 8, in *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, pp. 815–828, ACM.

Holgate, C. (2012), *LiveCode Mobile Development Beginner's Guide*, Packt Publishing, Limited.

Ishii, H., and B. Ullmer (1997), Tangible bits, in *the SIGCHI conference*, pp. 234–241, ACM Press, New York, New York, USA.

Johnston, W. M., J. Hanna, and R. J. Millar (2004), Advances in dataflow programming languages, *ACM Computing Surveys (CSUR)*, *36*(1), 1–34.

Jordà, S. (2003), Sonigraphical instruments: from FMOL to the reacTable, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pp. 70–76.

Jordà, S. (2004), Instruments and Players: Some Thoughts on Digital Lutherie, *Journal of New Music Research*, *33*(3), 321–341.

Jota, R., A. Ng, P. Dietz, and D. Wigdor (2013), How fast is fast enough?: a study of the effects of latency in direct-touch pointing tasks, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2291–2300, ACM, ACM.

Kammer, D., F. Lamack, and R. Groh (2010), Enhancing the expressiveness of fingers: multi-touch ring menus for everyday applications, in *AmI'10: Proceedings of the First international joint conference on Ambient intelligence*, Springer-Verlag.

Kim, K. J., S. S. Sundar, and E. Park (2011), The effects of screen-size and communication modality on psychology of mobile device users, in *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pp. 1207–1212, ACM.

Krueger, M. W., T. Gionfriddo, K. Hinrichsen, M. W. Krueger, T. Gionfriddo, and K. Hinrichsen (1985), *VIDEOPLACE—an artificial reality*, vol. 16, ACM.

Kurtenbach, G. P., A. J. Sellen, and W. A. S. Buxton (1993), An empirical evaluation of some articulatory and cognitive aspects of marking menus, *Human-Computer Interaction*, *8*(1).

Lamb, R., and A. Robertson (2011), Seaboard: a new piano keyboard-related interface combining discrete and continuous control, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pp. 503–506, Oslo, Norway.

Larkin, J. H., and H. A. Simon (1987), Why a Diagram is (Sometimes) Worth Ten Thousand Words, *Cognitive Science*.

Leavitt, J., S. John, A. Pressman, Z. Seuberling, and K. Lo (2013), Hopscotch, Available online at: `http://www.gethopscotch.com/`, retrieved July 1, 2013.

Lee, S., and S. Zhai (2009), The Performance of Touch Screen Soft Buttons, in *CHI 09' Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

Levenshtein, V. I. (1966), Binary codes capable of correcting deletions, insertions, and reversals, *Soviet Physics Doklady*, *10*(8), 707–710.

Levin, G., and Z. Lieberman (2005), Sounds from shapes: audiovisual performance with hand silhouette contours in the manual input sessions, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, pp. 115–120, National University of Singapore, Singapore, Singapore.

Li, S., T. Xie, and N. Tillmann (2013), A comprehensive field study of end-user programming on mobile devices, in *2013 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 43–50, IEEE.

Linden Research, Inc. (2012), Creatorverse.

Linden Research, Inc. (2013), Blocksworld.com.

Lundgren, S., and M. Hjulström (2011), Alchemy : dynamic gesture hinting for mobile devices, in *the 15th International Academic MindTrek Conference*, pp. 53–60, ACM, New York, New York, USA.

Luo, Y., and D. Vogel (2014), Crossing-based selection with direct touch input, in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pp. 2627–2636, ACM.

MacKenzie, I. S. (1992), Fitts' law as a research and design tool in human-computer interaction, *Human-Computer Interaction*, *7*(1), 91–139.

MacKenzie, I. S., and W. Buxton (1992), Extending Fitts' law to two-dimensional tasks, in *CHI '92: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 219–226, ACM Request Permissions, New York, New York, USA.

MacKenzie, I. S., A. Sellen, and W. A. S. Buxton (1991), A comparison of input devices in element pointing and dragging tasks, in *CHI '91: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 161–166, ACM Request Permissions, New York, New York, USA.

MacLaurin, M. B. (2011), The design of kodu: a tiny visual programming language for children on the Xbox 360, in *POPL '11: Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM.

Mayer, M., and V. Kuncak (2013), Game programming by demonstration, in *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pp. 75–90, ACM.

McDirmid, S. (2011), Coding at the speed of touch, in *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, pp. 61–76, ACM.

McLean, A., D. Griffiths, N. Collins, and G. Wiggins (2010), Visualisation of live code, in *EVA'10: Proceedings of the 2010 international conference on Electronic Visualisation and the Arts*, British Computer Society.

McPherson, A. (2012), Touchkeys: Capacitive multi-touch sensing on a physical keyboard, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, edited by G. Essl, B. Gillespie, M. Gurevich, and S. O'Modhrain, University of Michigan, Ann Arbor, Michigan.

McPherson, A., and Y. Kim (2010), Augmenting the acoustic piano with electromagnetic string actuation and continuous key position sensing, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Sydney, Australia.

McPherson, A., and Y. Kim (2011), Multidimensional gesture sensing at the piano keyboard, in *Proceedings of the 2011 annual conference on Human factors in computing systems*, pp. 2789–2798, ACM.

Microsoft (2013), Kinect.

Milanesi, C., L. Tay, R. Cozza, R. Atwal, T. H. Nguyen, T. Tsai, A. Zimmermann, and C. K. Lu (), Forecast: Devices by Operating System and User Type, Worldwide, 2010-2017, 2Q13 Update | 2524916, Gartner Inc.

Miranda, E., and M. Wanderley (2006), *New Digital Musical Instruments: Control and Interaction Beyond the Keyboard*, A-R Editions, Inc., Middletown, Wisconsin.

Moog, R., and T. Rhea (1990), Evolution of the keyboard interface: The bösendorfer 290 se recording piano and the moog multiply-touch-sensitive keyboards, *Computer Music Journal*, *14*(2), 52–60.

Moog, R. A. (1982), A Multiply Touch-Sensitive Clavier for Computer Music Systems, in *Proceedings of the International Computer Music Conference (ICMC)*, pp. 601–605.

Nacenta, M. A., P. Baudisch, H. Benko, and A. Wilson (2009), Separability of spatial manipulations in multi-touch interfaces, in *GI '09: Proceedings of Graphics Interface 2009*, Canadian Information Processing Society.

Norman, D. A. (2010), Natural user interfaces are not natural, *interactions*, *17*(3), 6–10.

Norman, D. A., and J. Nielsen (2010), Gestural Interfaces: a Step Backward in Usability, *interactions*, *17*(5).

Odowichuk, G., S. Trail, P. Driessen, W. Nie, and W. Page (2011), Sensor Fusion: Towards a Fully Expressive 3D Music Control Interface, in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim)*, pp. 836–841, IEEE.

Oehl, M., C. Sutter, and M. Ziefle (2007), Considerations on Efficient Touch Interfaces – How Display Size Influences the Performance in an Applied Pointing Task, in *Human Interface and the Management of Information. Methods, Techniques and Tools in Information Design*, pp. 136–143, Springer Berlin Heidelberg, Berlin, Heidelberg.

O'Modhrain, S. (2011), A framework for the evaluation of digital musical instruments, *Computer Music Journal*, *35*(1), 28–42.

Pane, J. F., B. A. Myers, and L. B. Miller (2002), Using HCI techniques to design a more usable programming system, in *Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on*, pp. 198–206.

Panic, I. (2012), Diet coda.

Paradiso, J. A. (1997), Electronic music: new ways to play, *IEEE Spectrum*, *34*(12), 18–30.

Pinch, T. J., and F. Trocco (2004), *Analog days: The invention and impact of the Moog synthesizer*, Harvard University Press, Cambridge.

Puckette, M. (2002), Max at Seventeen, *Computer Music Journal*.

Puckette, M., et al. (1996), Pure data: another integrated computer music environment, *Proceedings of the Second Intercollege Computer Music Concerts*, pp. 37–41.

Raptis, D., N. Tselios, J. Kjeldskov, and M. B. Skov (2013), Does size matter?: investigating the impact of mobile phone screen size on users' perceived usability, effectiveness and efficiency., in *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services*, pp. 127–136, ACM.

Reas, C., and B. Fry (2006), Processing: programming for the media arts, *AI & Society*, *20*(4), 526–538.

Reimer, P., A. Branzan Albu, and G. Tzanetakis (2011), Raydiance: A tangible interface for teaching computer vision, in *Advances in Visual Computing*, *Lecture Notes in Computer Science*, vol. 6939, edited by G. Bebis, R. Boyle, B. Parvin, D. Koracin, S. Wang, K. Kyungnam, B. Benes, K. Moreland, C. Borst, S. DiVerdi, C. Yi-Jen, and J. Ming, pp. 259–269, Springer Berlin Heidelberg, doi:10.1007/978-3-642-24031-7_26.

Resnick, M., et al. (2009), Scratch, *Communications of the ACM*, *52*(11), 60.

Roeber, H., J. Bacus, and C. Tomasi (2003), Typing in Thin Air: the Canesta Projection Keyboard - a New Method of Interaction with Electronic Devices, *CHI EA '03: CHI '03 Extended Abstracts on Human Factors in Computing Systems*.

Rogers, K., et al. (2014), Piano: Faster piano learning with interactive projection, in *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, pp. 149–158, ACM.

Rovan, J., M. Wanderley, and S. Dubnov (1997), Instrumental gestural mapping strategies as expressivity determinants in computer music performance, in *KANSEI-The Technology of Emotion*.

Saëns, S. (2011), Codea.

Sasangohar, F., I. S. MacKenzie, and S. D. Scott (2009), Evaluation of Mouse and Touch Input for a Tabletop Display Using Fitts' Reciprocal Tapping Task, *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, *53*(12), 839–843.

Schönauer, C., K. Fukushi, A. Olwal, H. Kaufmann, and R. Raskar (2012), *Multimodal motion guidance: techniques for adaptive and dynamic feedback*, ACM.

Scott, B., and V. Conzola (1997), Designing Touch Screen Numeric Keypads: Effects of Finger Size, Key Size, and Key Spacing, *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*.

Shneiderman (1983), Direct Manipulation: A Step Beyond Programming Languages, *Computer*, *16*(8), 57–69.

Slany, W. (2012), A Mobile Visual Programming System for Android Smartphones and Tablets, in *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 265–266.

Sodhi, R., H. Benko, and A. Wilson (2012), LightGuide: Projected Visualizations for Hand Movement Guidance, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 179–188, ACM, ACM Press, New York, New York, USA.

Soukoreff, R. W., and I. S. MacKenzie (2004), Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI, *International Journal of Human-Computer Studies*, *61*(6), 751–789.

Sutherland, I. E. (1964), Sketch Pad a Man-Machine Graphical Communication System, in *DAC '64 Proceedings of the SHARE Design Automation Workshop*, pp. 6.329–6.346, ACM Press, New York, New York, USA.

Takegawa, Y., T. Terada, and M. Tsukamoto (2011), Design and implementation of a piano practice support system using a real-time fingering recognition technique., in *Proceedings of the International Computer Music Conference (ICMC)*, pp. 387–394, University of Huddersfield, UK.

Terrenghi, L., D. Kirk, A. Sellen, and S. Izadi (2007), Affordances for manipulation of physical versus digital media on interactive surfaces, in *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1157–1166, ACM, New York, New York, USA.

Tillmann, N., M. Moskal, J. de Halleux, and M. Fahndrich (2011), TouchDevelop: programming cloud-connected mobile devices via touchscreen, in *ONWARD '11: Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, ACM.

Vermeulen, J., K. Luyten, E. van den Hoven, and K. Coninx (2013), Crossing the bridge over Norman's Gulf of Execution: revealing feedforward's true identity, in *CHI '13: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, p. 1931, ACM, New York, New York, USA.

Vogel, D., and P. Baudisch (2007), Shift: A Technique for Operating Pen-Based Interfaces Using Touch, in *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.

Vogel, D., and G. Casiez (2012), Hand occlusion on a multi-touch tabletop, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2307–2316, ACM, ACM Press, New York, New York, USA.

Wagemans, J., J. Elder, M. Kubovy, S. Palmer, M. Peterson, M. Singh, and R. von der Heydt (2012), A Century of Gestalt Psychology in Visual Perception: I. and II. , *Psychological Bulletin*, *138*(6), 1172–1217, 1218–1252.

Wagner, J., E. Lecolinet, and T. Selker (2014), Multi-finger Chords for Hand-held Tablets: Recognizable and Memorable, in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*, pp. 2883–2892, ACM.

Wanderley, M. M., and P. Depalle (2004), Gestural control of sound synthesis, in *Proceedings of the IEEE*, pp. 632–644.

Wanderley, M. M., and N. Orio (2002), Evaluation of Input Devices for Musical Expression: Borrowing Tools from HCI, *Computer Music Journal*.

Wilcox, E. M., J. W. Atwood, M. M. Burnett, J. J. Cadiz, and C. R. Cook (1997), Does continuous visual feedback aid debugging in direct-manipulation programming systems?,

in *CHI '97: Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pp. 258–265, ACM, New York, New York, USA.

Wolber, D., H. Abelson, E. Spertus, and L. Looney (2011), *App Inventor*, O'Reilly Media.

Yang, Q., and G. Essl (2012), Augmented piano performance using a depth camera, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Ann Arbor.

Yang, Q., and G. Essl (2013), Visual Associations in Augmented Keyboard Performance, in *NIME'13*, Deajeon, Korea.

Yang, Q., and G. Essl (2014), Evaluating gesture-augmented keyboard performance, *Computer Music Journal*, *38*(4), 68–79.

Yoo, M., J. Beak, and I. Lee (2011), Creating Musical Expression using Kinect, in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Oslo, Norway.

Zhai, S., P. O. Kristensson, C. Appert, T. H. Andersen, and X. Cao (2012), Foundational Issues in Touch-Screen Stroke Gesture Design - An Integrative Review, *Foundations and Trends in Human-Computer Interaction*, *5*(2), 97–205.

Zickuhr, K., and L. Rainie (2014), Younger Americans' Reading Habits and Technology Use.

Zorn, O. (2012), Pythonista.